Lecture 15: Training and Search for Speech Recognition

In earlier lectures we have seen the basic techniques for training and searching HMMs. In speech recognition applications, however, the networks are often so large that special techniques must be developed to handle them. For example, consider a speech system based on triphone, each one captured by a 7 node HMM. This is in fact a realistic model. Figure 1 shows the HMM template used ion the Sphinx system, If we assume that the average length of word in a English lexicon is 5 phonemes, and we have a 20,000 word vocabulary, the fully expanded HMM built from a bigram language model expand into words, which are then expanded into triphones, would contain 20,000*5*7 = 700,000 nodes. And with a full trigram model, we'd need $20{,}000^2$*5*7 (= 1.4e10) nodes! The Viterbi algorithm is impractical for networks this size, and the amount of training data we would need to train it would be staggering. This chapter considers some special techniques that have been developed for handling such networks.

We will first consider search methods, as they then will be used in the training algorithms.

1. Searching Speech-based HMMs

In a previous class we explored the beam-Viterbi and A* (or stack Decoding) algorithms for efficiently searching the only the most promising tasks. Both these algorithms work by maintaining a list of the most promising hypotheses (consisting of the best partial paths found so far), called the **agenda**. In general, they operate by choosing the highest ranked hypothesis from this set and exploring promising extensions, which in turn produce new hypotheses to add to the agenda. While there are many ways to organize this search, let us assume a word based model for the sake of illustration. In this scheme, a hypotheses would consist of a sequence of words found so far, their positions in the input, the probability of the best path in the HMM that generates these words, and the heuristic function that provides an estimate of how much work is left to complete the utterance. The agenda Items are ranked by a measure that combines the path probability and the hueristic score. Thus, to extend a hypothesis, we need to predict and evaluate various words that could come next. To extend this hypothesis we need to both add a

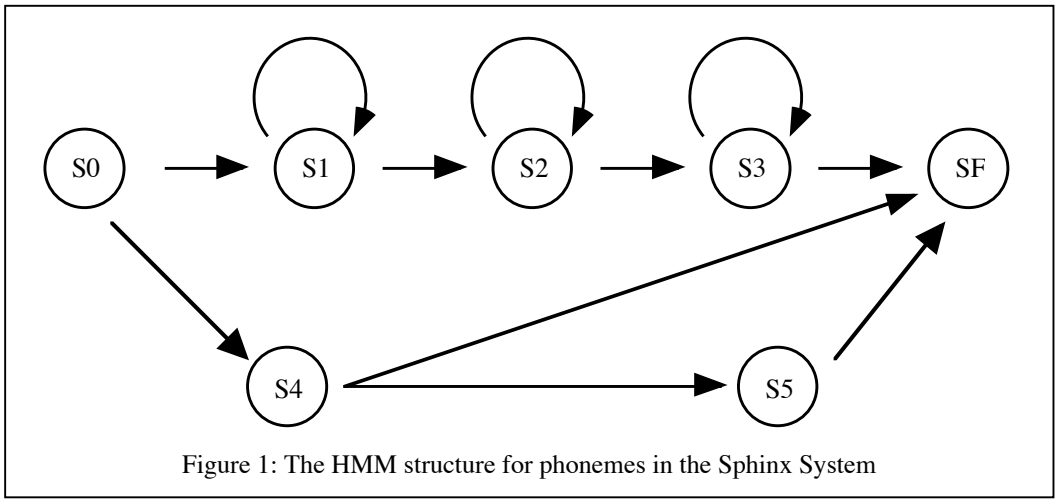

Figure 1: The HMM structure for phonemes in the Sphinx System

Hypothesis to Extend
(1, CAN, 15) (16, I, 20) (21,TAKE, 35)
Possible Extensions:
(36, A, 40), (36, A, 41), …, (36, A, 56)
(36, AN, 40), (36, AN, 41), …, (36, AN, 56)
(36, ANOTHER, 40), (36, ANOTHER, 41), …, (36, ANOTHER, 56)
(36, ANY, 40), (36, ANY, 41), …, (36, ANY, 56)
…..

Figure 2: Possible extensions to a Hypothesis "Can I take"

word to the sequence and how much of the input signal this new word covers. Thus there are many possible new hypotheses. An example is shown in Figure 2. Here we have a hypothesis that the word CAN occurs between time steps 1 and 15, then I to time step 20 and TAKE to time step 35. If we assume that words are never shorter than 4 time steps or longer than 20 time steps, we still have 16K possible extensions to consider for this hypothesis, where K is the size of the vocabulary. Clearly, just exploring one hypothesis to find some promising extensions could be a very expensive process!!

To make this more efficient, we need to have a way to identify promising hypotheses without having to search all of them. The first thing we can do is use the language model to select promising next words. For instance, we might just pick the 100 words that have the highest bigram probabilities P(w | TAKE). But it would be good to also combine this with an acoustic score as well, but without the expense of running the full recognizer. This motivates the need for a much faster, but less accurate, technique to identify promising hypotheses based on the acoustic input. This are called *Fast matching Algorithms*.

Fast Matching

There are several properties we would like of a fast matching algorithm. First, as just mentioned, the match must be accomplished quickly. Second, we would like the algorithm not to miss any good hypotheses, We can define a notion of being **admissible** (the same term as used for A* search): A fast match is admissible if it never undervalues the hypothesis that ends up being the best. As with the A* algorithm, we can show that any fast match algorithm that *overestimates* the actual probability (i.e., underestimates the cost) is admissible.

One effective technique is to develop a fast-match model for each word by not worrying about what state we are and simply using the maximum probability that could be obtained from any state. This would allow to to estimate the output probabilities for a sequence of code ook symbols without having to search the HMM. In particular, for each value c in the codebook, we can find the upper bound on the output probabilities from any state in the word HMM H:

$$UB(output_H(c)) = Max_{s \text{ in } HMM_w} P(c \mid s)$$

Output	S	U	C	V
S1	.01	.83	.15	.01
S2	.01	.02	.37	.6

Model for /S

Output	S	U	C	V
S1	.01	.01	.37	.61
S2	.01	.01	.07	.91

Model for /AE/

Output	S	U	C	V
S1	.02	.39	.49	.01
S2	.4	.1	.49	.01

Model for /D/

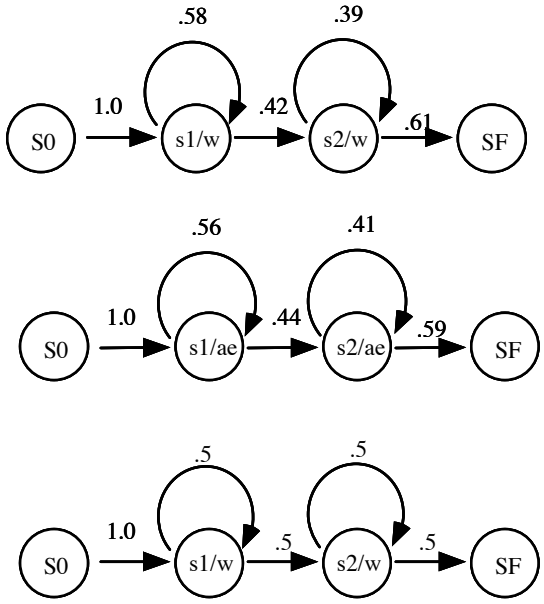

Figure 3: Three Phoneme HMM Models

Furthermore, for each reasonable path length, i, we can pre-compute the upper bound on any path of that length through the word HMM by computing the highest probnability path of that length:

$UB(path_H(i)) = Max_S P(S_{1,i})$, where $S_{1,i}$ ranges over all state sequences of length i

Now, the fastmatch estimate that a word w with HMM H starts at time k and ends in t time steps is

$FastMatch(w, k, t) = UB(path_H(t)) * \prod_{I=k, k+t} UB(output_H(o_i))$

It is simple to show that this is an upper bound on the probability of the best path though the HMM for the codebook sequence between times k and k+t. Furthermore, it can be computed in just t table lookups for each word. Thus we can use this to quickly estimate the probability of each word starting at the next time step, and select the best ones as new hypotheses to add to the search. Experience has shown that this technique is quite effective and coming up with a good set of candidates to consider in the search.

Consider an example of fast matching, but to keep the example simple we apply to phonemes rather than words. Say we have the three HMMS shown in Figure 3. Then we have the upper bounds on the output probabilities shown in Figure 4 and the upperbounds on transitions shown in Figure 5. The output probabilities are computed simply by taking the maximum output probability over all the states in the HMM. The best paths would be computed by using the Viterbi algorithm over the network ignoring the outputs.

Consider we are faced with the input UUCUUC. Looking just at alternates of length 3, we'd get the estimates from the fast matches shown in Figure 5.

UB(Output(c))	/S/	/AE/	/D/
S	.01	.01	.4
U	.83	.01	.39
C	.37	.37	.49
V	.6	.91	.01

Figure 3: The Upper bounds on the output probabilities

Node	Length 3	Length 4	Length 5	Length 6
/S/	.086	.05	.029	.017
/AE/	.091	.046	.026	.014
/D/	.063	.031	.015	.008

Figure 4: The Upper Bounds on Paths of length 3 through 6

.

HMM	Path Prob, length 3	Output U	Output U	Output C	UB estimate
/S/	.086	.83	.83	.37	.022
/AE/	.091	.01	.01	.37	3e-6
/D/	.063	.39	.39	.49	.005

Figure 5: The Fast Match Estimates for UUCUUC with Length 3

Thus, we predict that /S/ is the most likely phoneme consisting of the first 3 symbols, with /D/ and reasonable second guess and /AE/ very unlikely.

The exact same technique applies in the word case. The HMMS are just larger and more complex. But note the computation of the upper bounds can be done offline and stored in tables, making the computation at run time very efficient.

2.: Multi-pass Approaches to Allow More Complex Language models

In the above example, we used a bigram model. Clearly, to get more accurate speech recognition, we'd like to use a more powerful language model, but the expansion in the number of nodes to represent a language model such as a trigram is prohibitive, let alone for even more expressive models. So we need some additional techniques to allow such models to be used to improve the speech recognition.

The most basic approach is to not modify the speech recognition at all. We use a bigram model and then output a series of good hypothesis. This is called an **N-best** output. For instance, let us assume obtain the 7 best hypotheses from some input as shown in Figure 6. We can now **rescore** these hypotheses using some other language model. For instance, we might apply a trigram model to each and order them by the scores it produces. For instance, it might be that the trigram "WAKE THE ASPIRIN" is very rare and so the fourth best bigram hypothesis might drop to be sixth in the trigram. And similarly, "WAKE AND ASK" might be fairly rare and thus the bets bigram hypothesis might drop

1. CAN I WAKE AND ASK THEM 2. CAN I TAKE THE ASPIRIN 3. CAN I TAKE AN ASPIRIN 4. CAN I WAKE THE ASPIRIN 5. CAN I SHAKE THE ASPIRIN 6. CAN I SHAKE AND ASK THEM 7. CAN I SHAKE IT AND ASHPAN	1. CAN I TAKE THE ASPIRIN 2. CAN I TAKE AN ASPIRIN 3. CAN I SHAKE THE ASPIRIN 4. CAN I WAKE AND ASK THEM 5. CAN I SHAKE AND ASK THEM 6. CAN I WAKE THE ASPIRIN 7. CAN I SHAKE IT AND ASHPAN
An N-best Output with bigram model	The rescored list using a trigram

Figure 6: Rescoring using a Trigram model

to fourth in the trigram model. You might note that because the speech recognizer is not involved in the rescoring, we would use much more elaborate rescoring methods as well. We could run a probabilistic parser over them and rank them on grammaticality, or apply a semantic model as well and eliminate such semantic impossibilities as waking objects such as aspirin. Finding the most effective rescoring techniques remains an active research area.

The above methods are limited by the number of hypotheses one can realistically handle coming out of the speech recognizer. There are other techniques that implement rescoring within the speech recognizer itself that work directly on the hypothesis space (often called the **word lattice**) generated during recognition (which encodes vastly more alternatives that realistically captured in n-bets lists. This technique involves dividing the recognition into two steps:

1. we first do search forward through the observation sequence using a bigram model to produce a lattice of hypotheses;
2. we then do an A* algorithm over the lattice using the more complex language model.

In a number of systems, the second stage of this process searches backwards through the hypotheses and is referred to as the **forward-backward algorithm**. It is important to not confuse this technique with the Baum-Walsh Re-estimation procedure which is sometimes also referred to informally with the same name.

There is a reason why we search backwards rather than forwards. Remember for the A* algorithm we need to define a heuristic function. When searching backwards, we actually have available very good estimates for this. Since we are working backwards, recognizing the words in backward fashion from the end of the sequence top the beginning, we want H(h) to be an estimate of how likely our current hypothesis can be extended backwards to the beginning. But this is just the forward probability up to the node that starts the earliest word in the hypothesis. From the Viterbi first pass, we already know the probability of the maximum path to this node, which usually turns out to be a good approximation of the forward probability. This technique enables systems that still retain real-time performance yet can take advantage of richer language models.

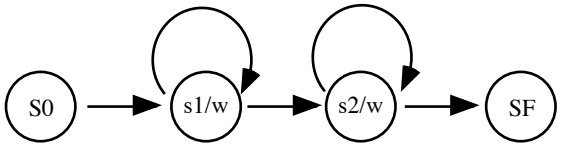

Figure 7: A simple phoneme model

3. Training Large HMMs For Speech

Now we have some effective ways to search these large networks, we can focus on issues relating to training them. Let us assume a phoneme based model for the sake of illustration. If we were to train HMM models for each phoneme in the way we saw earlier, we would need a set of labeled data upon which to train. But it is extremely time consuming to label the phoneme boundaries accurately in speech, even with the best automated tools. Often, there is no principled reason why a boundary is in one place rather than another. Without good training data, we won't be able to build accurate models, so the subword models that deal in units smaller than the syllable would appear very hard to train. Luckily, this is not the case. We can adapt the training algorithms we have already developed in order to automatically identify the subword unit boundaries that produce good recognition performance.

As with the Baum-Welsch reestimation procedure described earlier, we start with an arbitrary model and iterate to improve it. All we need to start this process are transcriptions of utterances, and we don't even need to know the word boundaries.

Let's explore this in a very simple case: we have an utterance consisting of a single word, and we want to use it to train models for the phonemes in the word. Consider the word *sad*, which consists of the phoneme string /s/ /ae/ /d/. Let's assume we will use the very simple 2-state HMM model shown in Figure 7 for each phoneme and assume that the codebook sequence (using our 4 element code book) we wish to train on is UUUCVCVVCUSC.

We then start training by segmenting the speech signal for the utterance into equal sized units so that we end up with one unit for each phoneme. With this segmentation, we run the Baum-Walsch procedure a few times to train each of the phoneme models. With these models in hand, we then build an utterance (in this case word) model by concatenating the phoneme models and use the Viterbi algorithm to find the best path through the HMM on the same input, which may give us a different set of phoneme boundaries. This new segmentation should better reflect the acoustics of the signal. We then run the Baum-Walch Re-estimation procedure to train the phoneme models again, this time using the new segmentation. We can iterate this entire procedure until we find that we are getting little change in the segmentation.

Consider our example. Say our initial segmentation divides the signal up equally: for /s/ we have UUUC, for /ae/, VCVVC and for /d/ CUSVC. With a little smoothing and a couple of iterations of Baum-Walsh, we obtain models for the phonemes shown earlier in Figure 1. Using these models, we now construct an HMM for the entire word *sad* by concatenating the models together as shown in Figure 8. We now run use the Viterbi algorithm to find the most likely path through this network on the training sequence, and obtain the state sequence:

S0, S1/w, S1/w, S2/w, S1/ae, S1/ae, S1/ae, S2/ae, S2/ae, S1/d, S2/d, S2/d, S2/d, SF.

From this sequence, we get a new segmentation, namely UUU for /S/, CVCVV for /AE/ and CUSC for /D/. We then retrain the phoneme models with this new sequence and get better models. In this simple example, we don't see any change on the second iteration. While this example is very simple, the algorithm actually works fairly well in practice, rapidly converging to quite reasonable segmentations of speech. Furthermore, in areas where the algorithm performs poorly, it produces results that are way off and fairly easy to detect and correct by hand.

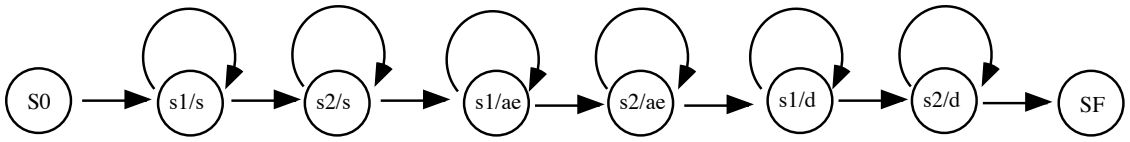

Figure 8: The models for *sad*

Of course, we usually would be working with extended utterances in a corpus rather than an isolated word. But the exact same technique works. We first compute the number of phonemes based on the transcription of the utterance and then divide the speech signal into equals, one part for each phoneme. We then train each phoneme as before, but now notice that we may have many instances of a single phoneme in an utterance, For instance, the utterance *Congress condemned bad art* has the phonemic sequence /k/ /ao/ /n/ /g/ /r/ /eh/ /s/ /k/ /ao/ /n/ /d/ /eh/ /m/ /d/ /b/ /ae/ /d/ /aa/ /r/ /t/. The phoneme /k/ occurs twice, as does /ao/, /eh/ and /d/. We would use each one of these instances to train a single model for the phoneme. Once the phoneme models are built, we can construct an HMM as before, and then run the Viterbi algorithm to obtain a better segmentation.

To give a better feeling for realistic models used for phoneme/triphone models, Figure 4 shows an HMM model that has proved to be quite successful in actual speech recognition systems such as the SPHINX system.