15.1 Language Modeling

15.1.1 Results of Google search homework

The class was asked to find the smallest number of words, which google could not find in a phrase search. A few two-word phrases not found by Google were:

- perplexity banana
- fish xenon
- avocado transistor
- snoopy meows
- isomorphic joystick
- radius flu
- zit ax

The most common element among the pairs is a food item paired with a technology-related item. “Snoopy meows” is particularly interesting in that it has two animal-related words. “Zit ax” may be the smallest number of letters forming real words that google does not find. The relative ease of finding two-word phrases not contained in the Google database illustrates the rapidity with which n-gram word models begin to fail (by falsely assigning zero probabilities), when one attempts to find statistics on the occurrence of several-word-long phrases.

15.1.2 Information Theory Review

Entropy is a measure of how much information a symbol stream contains, or how much, on average, an observer would be “surprised” by a result. It is defined as:

$$H(p) \triangleq -\sum_{i} p_i \log p_i$$

Intuitively, the negative log probability of an event indicates the amount of surprise, or new information, gained from an occurrence of an event. The entropy is a probability-weighted average of the information per event.
Cross-entropy is the amount of surprise an observer would get from a symbol stream actually distributed according to \( p(x) \) if the observer incorrectly assumed the symbols to be distributed as \( q(x) \).

\[
H_q(p) \triangleq - \sum_i p_i \log q_i
\]

Cross-entropy \( H_q(p) \) is never less than the true entropy \( H(p) \). Following the surprise intuition, an observer could never decrease the amount of surprise by making incorrect assumptions about the symbol stream’s distribution. This is stated by Jensen’s inequality:

\[
- \sum_i p_i \log q_i \geq - \sum_i p_i \log p_i
\]

For a stationary ergodic source, emitting symbols \( w \) (e.g. words or phrases), drawn from a true distribution \( P \), the relative frequency of occurrence of a symbol will converge to its probability, so in the limit, \( p \) can be replaced as follows:

\[
H_Q(P) = \lim_{N \to \infty} - \frac{1}{N} \sum_{i=1}^{N} \log Q(w_i)
\]

Related to the cross entropy, is the Kullback-Leibler KL divergence, which is the most popular measure of difference between two distributions. The KL divergence is simply the difference between cross entropy and true entropy, or the amount of “extra” surprise an observer would experience to due assuming the wrong distribution.

\[
KL(P||Q) = \sum_i P(x_i) \log P(x_i) - \sum_i P(x_i) \log Q(x_i)
\]

\[
= \sum_i P(x_i) \log \frac{P(x_i)}{Q(x_i)}
\]

### 15.1.3 Perplexity

If we define a language model \( Q \), and an \( N \)-word text string \( w_{1:N} \), then \( Q(w_{1:N}) \) is the probability of that text string occurring, as estimated by the model. As such, it must sum to unity over all possible word sequences. That is:

\[
\sum_{w_{1:N}} Q(w_{1:N}) = 1, \forall N
\]

Perplexity gives a measure of language difficulty, and is defined as the inverse geometric mean of all word sequence probabilities:

\[
\text{ppl}_Q = \prod_{w_{1:N}} Q(w_{1:N})^{-1/N}
\]

The geometric mean of a set of \( N \) numbers gives the side length of an imaginary \( N \)-dimensional uniform polygon (i.e. all sides the same length) that has the same volume as a polygon with side lengths defined by the numbers in the set. In the context of language modeling, the perplexity of a language \( L \) (given a certain model) is the vocabulary size of a hypothetical language \( L' \) with all words equiprobable, such that \( L \) and \( L' \) would have the same amount of entropy per word, and thus would be equally difficult.

Perplexity is also two raised to the the cross entropy between the distribution \( Q \) defined by a language model, and the true distribution \( P \), or rather the distribution that occurs in the body of text used for evaluation. Starting with the result from the previous section, and assuming the body of text is large enough, we remove the limit.
Table 15.1: Perplexities of text from the Wall Street Journal for various language models. About 1 million words in the set, drawn from a vocabulary of 45.4K words.

<table>
<thead>
<tr>
<th>Model</th>
<th>Probability</th>
<th>Perplexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>unigram</td>
<td>$p(w_t)$</td>
<td>1386</td>
</tr>
<tr>
<td>bigram</td>
<td>$p(w_t</td>
<td>w_{t-1})$</td>
</tr>
<tr>
<td>trigram</td>
<td>$p(w_t</td>
<td>w_{t-1}, w_{t-2})$</td>
</tr>
<tr>
<td>4-gram</td>
<td>$p(w_t</td>
<td>w_{t-1}, w_{t-2}, w_{t-3})$</td>
</tr>
<tr>
<td>5-gram</td>
<td>$p(w_t</td>
<td>w_{t-1}, w_{t-2}, w_{t-3}, w_{t-4})$</td>
</tr>
<tr>
<td>6-gram</td>
<td>$p(w_t</td>
<td>w_{t-1}, w_{t-2}, w_{t-3}, w_{t-4}, w_{t-5})$</td>
</tr>
<tr>
<td>d2-bigram</td>
<td>$p(w_t</td>
<td>w_{t-2})$</td>
</tr>
<tr>
<td>d3-bigram</td>
<td>$p(w_t</td>
<td>w_{t-3})$</td>
</tr>
<tr>
<td>d4-bigram</td>
<td>$p(w_t</td>
<td>w_{t-4})$</td>
</tr>
<tr>
<td>d5-bigram</td>
<td>$p(w_t</td>
<td>w_{t-5})$</td>
</tr>
<tr>
<td>d(3-5)-4-gram</td>
<td>$p(w_t</td>
<td>w_{t-3}, w_{t-4}, w_{t-5})$</td>
</tr>
</tbody>
</table>

The perplexity of a language is entirely dependent on the model used to determine sentence probabilities. The simplest possible model would be to assume that all sequences are equally likely, which would make perplexity exactly equal to $|V|$, the vocabulary size.

Table 15.1 shows perplexities of a set of text taken from the Wall Street Journal, as measured by several different language models. We can see that for the first three words, it helps to consider more words in the probability estimation. Beyond that (i.e. for 4-grams and beyond), additional words do not help substantially. The table also shows that considering words separated by more distance helps less, which one would expect. For some combinations, such as the d5-bigram model, where probabilities are conditioned on the word that occurred five words back, the perplexity is actually higher than the unigram model. This is actually an artifact of the finite size of the training data. With infinite training data, a model would never do worse by considering more previous words, but it might not do much better. However, extending the model backwards, i.e. using a probability model $p(w_t|w_{t-k})$ where $k$ is large reduces the available amount of training data, because there are fewer $k$-length sequences when $k$ is larger.

None of these models utilizes context. Some advantage can be gained by using context, as we will see in the discussion of cache-based models. The use of context faces the same problem as the use of longer windows: more information is gained with which to estimate the probability, but fewer training samples are available from which to learn how to estimate the probability.

Tri-gram modeling has become the dominant method for estimating probabilities. We can start out with a count ratio

$$H_Q(P) = -\frac{1}{N} \sum_{i=1}^{N} \log Q(w_i)$$

$$2^{H_Q(P)} = 2^{-\frac{1}{N} \sum_{i=1}^{N} \log Q(w_i)}$$

$$= (2^{\sum_{i=1}^{N} \log Q(w_i)})^{-\frac{1}{N}}$$

$$= \left( \prod_{i=1}^{N} 2^{\log Q(w_i)} \right)^{-\frac{1}{N}}$$

$$ppl_Q = \left( \prod_{i=1}^{N} Q(w_i) \right)^{-\frac{1}{N}}$$ The log is a base-2 log
of sequences found in the training data:

\[ Q(w_t|w_{t-1}, w_{t-2}) = \frac{C(w_t, w_{t-1}, w_{t-2})}{C(w_{t-1}, w_{t-2})} \]

where \( C() \) is the number of occurrences, the count function. However, \( C(w_t, w_{t-1}, w_{t-2}) \) is still sparse. Many allowable tri-grams will not exist in the training data, as we saw in the Google experiment. Using the count ratio alone would result in legitimate phrases being assigned zero probability. Even if we had a sufficiently large training set, such that all possible tri-grams were contained, the table required to store the results would have \( |V|^3 \) entries, which would be impractical in general purpose systems, where \( |V| \) may be upwards of 100,000. There are many solutions to this problem, all of which involve some form of smoothing, or replacing the zero entries with some non-zero value.

### 15.1.4 Deleted Interpolation

To make the problem tractable, we assume that the training data contains at least one instance of every word. Consider a model of the form

\[
Q(w_t|w_{t-1}, w_{t-2}) = \lambda_3 f(w_t|w_{t-1}, w_{t-2}) + \\
= \lambda_2 f(w_t|w_{t-1}) + \\
= \lambda_1 f(w_t)
\]

\[
\lambda_1 + \lambda_2 + \lambda_3 = 1
\]

where \( f() \) is the count ratio described above. Under this model, every word with any occurrence in the training data would have a non-zero probability, because of the \( \lambda_1 \) term. The use of counts and “meta-counts”, i.e. counts of how many words have a certain count, is a common theme in language modeling. It seems that the important information about a word or structure is not what type of word it is or what it means, but how often it occurs.

The next question is how to train the \( \lambda \) values. An intuitively reasonable solution would be to use maximum likelihood (ML) training:

\[
(\lambda_1, \lambda_2, \lambda_3)^* = \arg\max_{(\lambda_1, \lambda_2, \lambda_3)} \sum_t \log Q(w_t|w_{t-1}, w_{t-2})
\]

Unfortunately, if we use the same data to train \( \lambda \) as we used to obtain the count ratios, then the optimal solution will always be \( \lambda_3 = 1; \lambda_2 = \lambda_1 = 0 \). The reason for this is that ML training will, in effect, try to match the predicted probabilities \( Q(w_t|w_{t-1}, w_{t-2}) \) to the distribution observed in the training data. With \( \lambda_{1,3} = [0, 0, 1] \), the match is perfect. Considering that the purpose of the \( \lambda \) coefficients is to mitigate the effect of the mismatch between the distribution manifested in the training data and the true distribution, it should not be surprising that the \( \lambda \) coefficients are mis-trained if they are trained on the original data, where there is no mismatch.

We can address this problem by dividing the training set into a kept set, and a held-out set. The count ratios \( f() \) are trained on the kept set, and then \( \lambda \) is is trained to optimize performance on the held-out set. The name for this type of smoothing, deleted interpolation, comes from the fact that some of the training data is deleted, and the remaining data is interpolated to yield a smooth distribution. Additionally, \( \lambda \) can be made to depend on the words via the counts. If \( C(w_2, w_1) \) is large, then \( \lambda_3 \) would get a high weight. Otherwise, more of the weight would go to \( \lambda_2 \). To simplify, we can work with only two coefficients at a time by combining coefficients:

\[
\lambda_3 = \lambda_3' \\
\lambda_2 = \lambda_2' \lambda_3' \\
\lambda_1 = \lambda_1' \lambda_3' \\
\lambda_1' + \lambda_2' = 1 \\
\lambda_3' + \lambda_4' = 1
\]
Then, \( p(w_3|w_2, w_1) = \lambda_3^I f(w_3|w_2, w_1) + \lambda_4^I (\lambda_2^I f(w_3|w_2) + \lambda_1^I f(w_3)) \)

We would like to use a word-dependent \( \lambda \), since different words interact differently with their context. For example, in the bi-gram case, we would have

\[ p(w_t|h_t) = \lambda(h_t) f(w_t|h_t) + (1 - \lambda(h_t)) f(w_t) \]

where \( h_t \) is history. Unfortunately, using a separate value of \( \lambda \) for each word would result in too many \( \lambda \)s, a problem both for storage and for training. So we can cluster words, based on their counts in the training data, assigning the same value of lambda to words with similar counts. We can summarize the procedure as follows:

- Divide the training data into kept and held out sets.
- Compute \( f(w_t|h_t) \) and \( f(w_t) \) under the kept set.
- Compute counts on held out set \( N(h_t) \)
- For each range \( R_t \), represented by \( r \in 1, 2, \ldots, N \), where \( N \) is the number of count ranges, find

\[ \lambda^I_c = \arg \max_{\lambda} \sum_{h, c(h) \in R_t} \sum_{w_t} p_H(w_t|h) \log[\lambda_c f(w_t|h) + (1 - \lambda_c) f(w_t|h)] \]

\[ p_H(w_t|h) = \frac{N(w_t, h)}{N(h)} \text{ From the held-out set} \]

Notice that this is fundamentally a procedure for minimizing the Kullback-Leibler divergence between the true underlying distribution of word sequences (at least as far as the true distribution is represented in the held-out set) and the distribution described by the language model. More generally, any Maximum-likelihood estimation technique will minimize the KL divergence between a parametric model distribution (e.g. a gaussian) and an observed distribution. This technique enjoys the computational benefit of not requiring iterative training. The optimal value of \( \lambda \) can be found by taking the gradient of the right-hand side of the previous equation, using a Lagrange multiplier [EW99], and computing the minimum directly.

### 15.1.5 Adaptive Language Modeling

As humans, we take for granted our ability to identify acoustically ambiguous words based on context. Adaptive language modeling is based on the idea that \( p(w_t|h_t) \) should not stay fixed, but should depend on context, and that previous occurrences of variations on a word (run vs. running) or similar words (e.g. jog) give us similar information about the context. As in any language model, the ideal is that \( p(w_t|h_t) \) be 1 when \( w_t \) is the correct word, very small for all other words. Cache-based language models are one approach.

\[ p(w_t|h_t) = \sum_i p_o(w_t|i)p(i|\phi(w_{t-1}), \phi(w_{t-2})) \]

\( \phi(w) \) class of word \( w \), clustering function

\( p_o(w_t|i) \) is a context specific language model, for class \( i \)

\( i \) is an integer associated with a certain context

\[ p_o(w_t|i) = \lambda_c p_{\text{cache}}(w_t|i) + (1 - \lambda_c) p_{\text{class}}(w_t|i) \]

\[ p_{\text{class}}(w_t|i) = \frac{c(w_t, i)}{C(i)} \]

\( p_{\text{cache}}(w_t|i) \) = a function that gives high probability for words seen recently in ith cache.
For example, let the \( i \)-th cache consists of the set of words \( S_i \), and \( k_t \) be a K-length history at time \( t \). Then

\[
p_{\text{cache}}(w_t|i) = \frac{|w : w \in k_t \cap S_i|}{K}
\]

The choice of how to decide which words go in a given set is an open question. One choice would be to use Ling decision trees. Another approach would be to minimize the entropy of each class, but there is no solution known to be optimal in all cases.

### 15.1.6 Backoff Language Modeling

When the training set includes many instances of a given history \( h_t \) we would expect the word distribution for that history \( p(w_t|h_t = h_t) \) to be a good estimate of the true underlying distribution. When that history occurs only a few times, however, the model distribution will be based on sparse data, and less accurate. Backoff language modeling is an attempt to use this intuition to adjust the distributions, and is one of the most widely used approaches to language modelling. When the count of a certain history higher-order history (e.g. the last two or three words) exceeds a threshold, a discounted maximum likelihood model is used. If the count is below threshold, \( p(w_t|h_t) \) is formed from lower order distributions. When implementing a backoff language model, it is important to keep all of the coefficients normalized, so that the resulting distributions sum to unity. The algorithm can be implemented recursively:

Define \( d_c(w_t, w_{t-1}, w_{t-2}) \) is the “discount” coefficient

and \( \alpha(w_{t-1}, w_{t-2}) \) is the backoff weight

\[
p_{BO}(w_t|w_{t-1}w_{t-2}) = \begin{cases} 
  d_c(w_t, w_{t-1}, w_{t-2}) \frac{c(w_t, w_{t-1}, w_{t-2})}{c(w_{t-1}, w_{t-2})} & \text{if } c(w_t, w_{t-1}, w_{t-2}) > \tau \\
  \alpha(w_{t-1}, w_{t-2}) p_{BO}(w_t|w_{t-1}) & \text{else}
\end{cases}
\]

The discount function \( d_c \) would be a function that approaches unity for large values of \( c(h_t) \). The \( \alpha \) term is chosen to ensure that the resulting distribution is legitimate, i.e. that it sums to unity over all possible word sequences. So we start with a higher-order model (considering a longer history), and then back off to progressively lower order models until we get enough instances of a history to provide a good estimate of the distribution. If \( d_c \) were always unity, then the backoff model would reduce to the maximum likelihood solution. The discount coefficient determines how much probability mass to take from the maximum likelihood solution and move to the lower-order probability estimate. The exact form of \( d() \) is an active research topic in language modeling [CG98]. The use of smoothing algorithms, which replace zeros in the distribution with non-zero values, is equivalent to the use of discount functions in backoff modeling. The major benefit of using a backoff model is that the \( p_{BO} \) table does not need to be full, since zeros will be filled in with lower-order estimates, which allows good distribution estimates to be built from relatively small training sets, and requires much less memory than an algorithm that requires a full lookup table.

There are many varieties of backoff language models, differing mostly in the choice of discount function \( d() \). Additive smoothing, which is equivalent to Laplace smoothing, is one example, wherein the word sequence counts are adjusted by a factor \( \delta \):

\[
d_c(w_t, w_{t-1}, w_{t-2}) p_{ML}(w_t|w_{t-1}, w_{t-2}) = \frac{C(w_t, w_{t-1}, w_{t-2}) + \delta}{C(w_{t-1}, w_{t-2}) + |V|\delta}
\]

If \( \delta = 0 \), additive smoothing reduces to the maximum likelihood solution (\( p_{ML}() \) is simply the ratio of counts). As \( \delta \) becomes large, the estimate is more extensively smoothed, and approaches a uniform distribution. Additive smoothing is straightforward and intuitive, but fails to take advantage of contextual information; the same degree of smoothing is always applied, regardless of context.

A better solution is known as Good/Turing smoothing [CG98], an algorithm that pretends that sequences with \( r \)
occurrences actually occurred $r^*$ times, where:

$$ r^* = (r + 1) \frac{n_{r+1}}{n_r} $$

where $n_r$ is the number of n-grams that occurred exactly $r$ times. The intuition is that probability mass is taken from sequences with high counts and given to sequences with lower counts. What qualifies as a high or low count depends on how many n-grams have those counts.

Katz smoothing extends this notion, by redistributing the counts from n-grams with non-zero counts to n-grams with zero counts, according to the ratios of the next lower-order distribution. For example, starting with bi-grams, non-occurring bi-grams would be assigned a smoothed count $C^*$ according to their unigram counts.

$$ C^*(w_{i-1}w_i) = \begin{cases} d_r r & \text{if } r > 0 \\ \alpha(w_{i-1})P(w_i) & \text{else} \end{cases} $$

where $P(w_i)$ is the probability based on unigram count ratios. The term $\alpha$ is chosen to ensure that the smoothed counts lead to a legitimate distribution. Kneser-Ney smoothing [HAH01], one of the most powerful known smoothing techniques, further extends the notion of meta-counts and meta-meta-counts.

When using any backoff smoothing technique, it is important to normalize the smoothed count ratios correctly to ensure that a legitimate probability distribution is obtained. Specifically, for a bigram backoff model, the following equality must be true:

$$ \sum_{w_t} p_{BO}(w_t|w_{t-1}, \ldots, w_{t-2}) = 1 \ \forall w_{t-1}, \ldots, w_{t-2} $$

By solving for the $\alpha$ required to generate a valid distribution, we get:

$$ \alpha(w_{t-2}, w_{t-1}) = \frac{1 - \sum_{w: c > \tau} d_c p_{ML}}{\sum_{w: c \leq \tau} p_{BO}(w_t|w_{t-1})} = \frac{1 - \sum_{w: c > \tau} d_c p_{ML}}{1 - \sum_{w: c \leq \tau} p_{BO}(w_t|w_{t-1})} $$

The first version is extremely expensive to compute, because it requires that the discount function be summed over every sequence with a count under the count threshold. Since there will be far fewer sequences with counts above the threshold, the second version is much easier to compute.

### 15.2 Large Vocabulary Systems

There are many different types of speech to which recognition techniques might be applied. In roughly increasing order of difficulty, these include:

- **Isolated Word or Phrase** For example, command/control applications, such as cell-phone dialers.
- **Read speech**
- **Lecture speech** Prepared speech, with few disfluencies.
Conversational Speech  Between acquaintances.

Meeting Speech

Group Speech  in an informal group.

Casual Conversational Speech  between close friends or family members.

For isolated word or phrase recognition, it is possible to use whole word models, where every allowable word or phrase has its own HMM. Each HMM can be trained separately using the E-M algorithm, making training relatively easy. Bayesian methods can then be directly applied to choose the most likely phrase given any utterance.

All of the other speech types can contain any combination of any words. This implies that Bayesian methods are not practical, since that would require a probability estimate for every possible word sequence. As humans, we know that language is structured; not all word combinations form reasonable phrases. In order to cope with large vocabularies, it is necessary to build recognition systems that take advantage of language’s structure.

Coarticulation is one example of the structure of speech. The pronunciation of a given phoneme varies depending on the phonemes surrounding it. As fluency and speed increase, it is increasingly the case that phoneme targets are approached, but not met. Recall the example from the second lecture of the phrases “in Paris” and “in the house.” The two ‘n’s are preceding two different sounds (specifically, two different places of articulation), and the sound of the two ‘n’s differs accordingly. Figure 1 shows an example.

![Figure 15.1: The pronunciation of phoneme 3, with articulation target T3, depends on the subsequent targets T4 and T4'. The actual pronunciation, P or P', falls somewhere between the target, T3, and the subsequent target, T4 or T4', respectively.](image)

The most general type of speech recognition, continuous conversational speech, with no topic constraints, between close friends, is fraught with difficulties. Even the word boundaries are difficult to locate, as the silence detector project demonstrated. Standard Bayes theory is less applicable, because the number of possible sequences is so large. For example, the number of five-word sentences drawn from a 100,000 word vocabulary would be $5^{100,000}$.

To deal with the problems posed by the large vocabulary recognition task, systems must explore multiple hypotheses simultaneously, and choose wisely what possibilities to consider, discarding improbable sequences as early as possible. The recognition task becomes similar to a search procedure. See [RN02] for more information on search techniques. Since it is not practical to have a separate model for each word, we must use submodels for parts of words, and share submodels between words.
15.2.1 Tri-phones

In the structure of HMMs, we already have a sort of subword model, the HMM hidden state, and its output distribution. Words can be decomposed into parts, such as syllables, phones, subphones (e.g. the beginning, middle and end of a phone), and phones or subphones in a given context. The most common decomposition is based on context-dependent phones. Three-state tri-phone models consider a phone along with the preceding and following phone, using three HMM states to model each phone (nine for the tri-phone). Considering more context results in a more accurate model of the N-phone, because the set of N-phones considered to be the same (e.g. having the same phone and the same context) is more similar. With less context, there are still coarticulation effects from phones just outside the context window that are not considered. However, the extra accuracy of additional context comes at the expense of having fewer training samples for each N-phone. For example, the most common 5-phone is less common than the most common tri-phone. Tri-phone models are the most common, but some labs (specifically IBM) use 5-phone models and have achieved improvements relative to tri-phone model. One can think of the use of more context phones as analogous to using higher order terms in a Taylor series: in both cases the effects die off gradually as we get farther and farther out.

It is important to recognize that different words phones share phones, and phones share states, we must still keep track of where we are in a word or phrase. So, for example, in an HMM modeling the word “banana,” there are three occurrences of the /ae/ sound. These might share output distributions, but the HMM would still assign a zero-probability to a transition from the first /ae/ directly to the end of the word.

Notation: x-y+z denotes phone x, with left (preceding) context of phone y and right (following) context of phone z. For example, the phrase “Beat it” would be phonetically transcribed as “sil b iy ti t sil.” The tri-phones would be:

- sil
- b-sil+iy
- iy-b+t
- t-iy+ih
- ih-t+t
- t-ih+sil
- sil

Tri-phones can be considered in a word-internal manner, where only phones from the same word are counted as part of the context. Alternately, cross-word tri-phones may be used. The use of only word-internal tri-phones implicitly denies the occurrence coarticulation across word boundaries, which we know to occur, but it eases the computational burden by segmenting the problem. It also requires less training data.

Tri-phones provide temporal context and a way to model coarticulatory effects. Additionally they add more states to an HMM, making it more powerful. However, sometimes many states are needed to accurately model speech, which can lead to problems. For example, using 45 phones would require $45^3 = 91125$ tri-phones, and with three states per phone, 273,375 HMM states, without even counting the requirements of the language model. Using 39 features (e.g. 13 MFCCs with delta and acceleration coefficients), and modeling each state’s output distribution as a mixture of 10 gaussians would result in 215 million parameters to estimate. Such a quantity is not practical for computers of today or the near future.
15.2.2 Parameter Tying

One solution to this parameter explosion is known as distribution tying. The basic idea is that different states can use the same distributions, if they share certain characteristics. Language modeling provides a couple of techniques for accomplishing this. If there are too few instances of a triphone a-b+c, output distributions could be estimated using a-b, or a+c, as in backoff language models. Smoothing can also be used; for example (recall that $x$ represents the output associated with state $a$):

$$p(x|a-b+c) = a_1 f(x|a-b+c) + a_2 f(x|a-b) + a_3 f(x|a)$$

Clustering provides a superior method for sharing parameters, by using a decision tree to choose which states should share output distributions. Clustering algorithms can be divided into top-down and bottom-up algorithms.

15.2.2.1 Bottom-up Clustering

HTK uses bottom-up clustering. This approach begins with all contexts considered distinct, and makes the problem manageable by using a weak acoustic model (e.g. a single Gaussian component per state). For tri-phones, the beginning, middle and end phones would be clustered separately.

After training the initial model for all contexts, the states with the two most similar output distributions are merged, and a new distribution is formed for the pair, using the following formula:

$$\mu_{ij} = \frac{1}{2}(\mu_i + \mu_j)$$

$$\sigma_{ij}^2 = \frac{1}{2}(\sigma_i^2 + \mu_i^2 + \sigma_j^2 + \mu_j^2) - \mu_{ij}$$

The symmetric Kullback-Leibler divergence is a common choice of distance measure used to determine the similarity between distributions.

$$d(i, j) = \frac{1}{2}[D(i||j) + D(j||i)]$$

Assuming the output distributions are diagonal-covariance gaussians, this reduces to

$$d(i, j) = \left[\frac{1}{n} \sum_{k=1}^{n} \frac{(\mu_{ik} - \mu_{jk})^2}{\sigma_{ik}\sigma_{jk}}\right]^{\frac{1}{2}}$$

The assumption of diagonal-covariance gaussians implies that individual features are not correlated with each other. This is not strictly guaranteed, but is in practice a reasonable assumption, because the discrete cosine transform (DCT), used in calculating MFCCs, tends to decorrelate its inputs.

After merging the two most similar states, the process repeats, until the number of states has been reduced to some acceptable small number, or until the distance between the two closest states is above some threshold. After the states are clustered, the single component gaussians are converted into mixtures of gaussians during EM training, so that the output distributions can be better modelled.

The bottom-up method requires that the individual gaussians be well-estimated at the beginning. But if there are many tri-phones (and there are), then there may not even be enough data to reliably estimate diagonal gaussians.
15.2.2.2 Top-down Clustering

Top down clustering provides a solution to this problem, using phonetic decision trees, similar to those we saw earlier in the context of mapping from base forms (phonemes) to surface forms (phones). This algorithm asks a series of questions (e.g. is the left phone a liquid?, is the right phone a nasal?) to group states sharing characteristics into clusters. At each leaf node (where there are no more questions to be asked), a gaussian mixture is used to model all phones that satisfied the series of questions leading to that leaf node.

Top-down clustering methods have several desirable qualities. Unlike bottom-up clustering, they are guaranteed to find some pcontext-dependent model. They will never back off all the way to no context. Top-down clustering also utilizes the heuristic knowledge of human phonetic experts to design the question set. Decision trees can be formed to provide each leaf node with an adequate amount of data from which to estimate the gaussian mixture parameters. They can even have context beyond just the preceding and following phones by incorporating questions about more distant context into the decision tree.

The obvious question, then, is how to construct the decision tree. In constructing base form to surface form maps, we used entropy minimization (i.e. we minimized the entropy of the distribution of a single leaf node. In the clustering problem, however, we need not only to accurately model the phone distributions, we also need to ensure there is adequate data at each leaf node. The maximum likelihood method, which approaches entropy minimization with infinite data and accurate models, is implemented by the following equation:

\[
H(x|q) = -\lim_{N \to \infty} \frac{1}{N} \sum_{i=1}^{N} \log p(x_i|q), x_i \sim p(x|q) \forall i
\]

References


