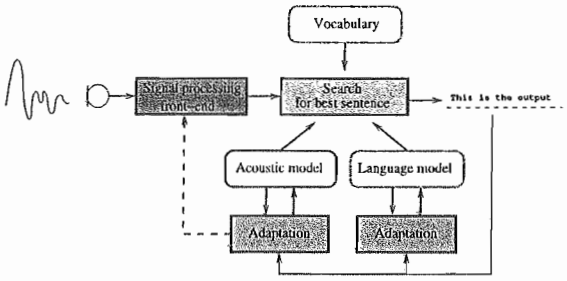
2008 IIB 4F11 SPEECH AND LANGUAGE PROCESSING

PROF PC WOODLAND

Solutions to 4F11 Speech and Language Processing, 2008

1. *Statistical Speech Recognition & HMM Training*

(a) The generic speech recogniser is



The primary models are the *acoustic* model and the *language* model. The language model assigns a probability $P(W)$ to sentence hypotheses W constructed from words in the *vocabulary* (which also defines the pronunciation in terms of the acoustic model units). The acoustic model assigns likelihood $p(O|W)$ to the acoustic observation sequence which is produced by the *front-end* (e.g. MFCCs). The search process finds the modt likely string. *Adaptation*, if used can be performed to tune the parameters of the acoustic and language models. [25%]

(b)(i) The forward probability is defined as

$$\alpha_j(t) = p(\boldsymbol{o}_1, \ldots, \boldsymbol{o}_t, x(t) = j|\lambda)$$

where $x(t)$ denotes the state occupied at time t, and λ represents the HMM parameter set.

The backward probability, $\beta_j(t)$, is defined as

$$\beta_j(t) = p(\boldsymbol{o}_{t+1}, \ldots, \boldsymbol{o}_T|x(t) = j, \lambda)$$

$\alpha_j(t)$ can be computed efficiently

$$\beta_j(t) = \sum_{i=1}^{N} \alpha_i(t-1)a_{ij}b_j(\boldsymbol{o}_t)$$

where $\alpha_i(0) = 1$ for $i = 1$ and zero otherwise. Recursion moves forwards in time.

$\beta_j(t)$ can be computed efficiently

$$\beta_j(t) = \sum_{i=1}^{N} a_{ji}b_i(\boldsymbol{o}_{t+1})\beta_i(t+1)$$

where $\beta_j(T+1) = a_{jN}$. Recursion moves backward in time. Note $p(\mathbf{O} = \beta_1(0)$. [25%]

(b)(ii)

$$\begin{aligned}\alpha_j(t)\beta_j(t) &= p(o_1 \ldots o_t, x(t) = j\,|\lambda)p(o_{t+1} \ldots o_T\,|x(t) = j, \lambda) \\ &= p(\mathbf{O}, x(t) = j\,|\mathcal{M}) \\ &= p(\mathbf{O}\,|\lambda)P(x(t) = j\,|), \lambda) \\ &= p(\mathbf{O}\,|\lambda)L_j(t)\end{aligned}$$

Hence, [10%]

$$L_j(t) = \frac{1}{p(\mathbf{O}\,|\lambda)}\alpha_j(t)\beta_j(t)$$

(c) A composite HMM is constructed for each sentence in the training corpus according to the word level transcription (allowing for optional silences), and the pronunciation dictionary. If there isn't a single pronunciation per word then the composite HMM has branches and either the *Viterbi algorithm* can be used to select pronunciation variants (given some initial models) or run forward-backward on the network. Note that this is also an issue for optional inter-word silences. Then the forward-backward algorithm is run on the complete sentence-level composite HMMs and suitable statistics for HMM training found. The HMMs can be initialised typically either by a flat start (all parameters equal, just define left-to-right topology) or by using some estimates of HMM parameters from a phone-level labelled set of data. [20%]

(d)(i)/(ii) Forward probability pruning. Calculate the maximum value of $\log \alpha_j(t)$ at each time and only consider models which have a maximum value $\log \alpha_j(t)$ within a threshold. The same method can be applied on the backwards pass to the $\log \beta_j(t)$ values. The state posterior is found from $\alpha_j(t)\beta_j(t)$ and a threshold on this from the maximum value gives a very tight beamwidth. In practice the $\alpha_j(t)$ pruning results in a beam a few words wide and the posterior pruning is then one or two phones wide. Note that this is very important for long utterances since only a few percent of the models can be active and greatly speeds training. [20%]

2. *Modifications for Speech Recogniser*

(a) MFCCs - take cosine transform of log filterbank energies (assuming that the filterbank is on a mel scale). Might take a vector of 24 energies and compute 12 MFCCs. If P channels and m_j is the energy of the jth channel then the MFCC component

$$c_n = \sqrt{\frac{2}{P}}\sum_{i=1}^{P} m_i \cos\left[\frac{n(i-\frac{1}{2})\pi}{P}\right]$$

This will reduce computation (smaller feature vectors) and storage. It will also increase accuracy as diagonal covariance matrices are a better approximation. [20%]

(b) It is proposed to add MFCC time differentials to the feature vector. Normally these are smoothed over several frames. They take account of the first-oder local dynamics to account (in part) for the poor HMM assumptions. The change will increase accuracy but make the feature vector two or three times as large and will increase computation (unless pruning is more effective due to increase in accuracy). [15%]

(c) The state distribution is not completely uncorrelated and may be non-Gaussian. Better to use a Gaussian mixture with

$$b_j(\mathbf{o}) = \sum_{m=1}^{M} c_{jm} b_{jm}(\mathbf{o}) = \sum_{m=1}^{M} c_{jm} \mathcal{N}(\mathbf{o}; \boldsymbol{\mu}_{jm}, \boldsymbol{\Sigma}_{jm})$$

c_{jm} is the component weight, or prior. For this to be a pdf it is necessary that

$$\sum_{m=1}^{M} c_{jm} = 1 \quad \text{and} \quad c_{jm} \geq 0$$

This increases computation and storage roughly proportional to the number of components but also reduces word error rate (if there is enough training data). [20%]

(d) Use a full covariance matrix. This allows the correlations between feature elements to be explicitly modelled but requires (for d dimensional data) a matrix with $d(d+1)/2$ parameters. if the feature vector has strong correlations and there is enough training data it will improve performance. Normal approach though is to do global decorrelation and use (a mixture) of diagonal covariance Gaussians which are more flexible.

The full covaraince matrix will greatly increase computation and storage (by a factor of about $d/2$ for the output probability caluculations. [20%]

(e) Cross-word triphones. Convert each monophone to a cross-word context-dependent phone model (model depends on the immediate left and right phone context as well as the phone itself and the context extends across word boundaries). It allows co-articulation to be explicitly modelled. This complicates training as the number of models is greatly increased and some type of smoothing or parameter sharing is reuqired. Here it is suggested that state-tying via phonetic decision trees are used. These group contexts so that models can be robustly estimated for the grouped contexts and unseen triphones can be dealt with. The phonetic decision tree is grown automatically from training in a top-down fashion with questions chosen so as to maximise an approximate likelihood of the training data. The questions for splits are chosen from a pool of linguistoc questions which yield generalisation ability.

For cross-word triphones (from monophones) decoding is greatly complicated and computational cost increases (and storgae since there will usually be far more parameters even with tying). It can have a large decrease in word error rate (factor of two or more). . [25%]

3. *N-gram Language Models and Weighted Finite State Networks*

(a) Some aspects of N-gram language models that make them useful for speech and language processing systems are: N-gram language models can be estimated from large amounts of text, which makes it possible to build models for specific conditions - tasks, genres, dialects, languages - if relevant data is available. The statistics needed for estimation of N-gram model parameters are easily gathered, simply by counting. N-gram models are predictive, or left-to-right, in nature, which makes them suitable for use with (for instance) time-synchronous Viterbi search in speech recognition. Three practical issues which arise are discounting, backoff, and pruning. Discounting strategies modify the maximum likelihood estimation procedure so that some probability mass is held aside for unseen events. Backoff strategies estimate use lower-order N-gram probabilities when higher-order N-grams are not observed (or infrequently observed) in the training text. Pruning strategies reduce the size of estimated N-gram model by discarding infrequently occurring or uninformative N-grams to reduce the memory required to store the model; pruning strategies trade off performance vs memory usage. [20%]

(b) The probability assigned to a word sequence $W = w_1 \dots w_K$ by a bigram language model is

$$P(W) = P(w_K|w_{K-1}) \cdots P(w_2|w_1)P(w_1)\,.$$

The predictive form of the language models is

$$P(W) = \prod_{k=1}^{K} P(w_k|w_1 \dots w_{k-1})$$

where w_0 is a null token, i.e. $P(w_1|w_0) = P(w_1)$. The assumption which underlies the bigram language model is that $P(w_k|w_1 \dots w_{k-1}) \approx P(w_k|w_{k-1})$ so that [20%]

$$P(W) = \prod_{k=1}^{K} P(w_k|w_{k-1})$$

(c) The stupid backoff scheme generates order k N-gram scores as follows

$$S(w_i|w_{i-k+1}^{i-1}) = \begin{cases} \frac{f(w_{i-k+1}^{i})}{f(w_{i-k+1}^{i-1})} & \text{if } f(w_{i-k+1}^{i}) > 0 \\ \\ \alpha\, S(w_i|w_{i-k+2}^{i-1}) & \text{otherwise} \end{cases}$$

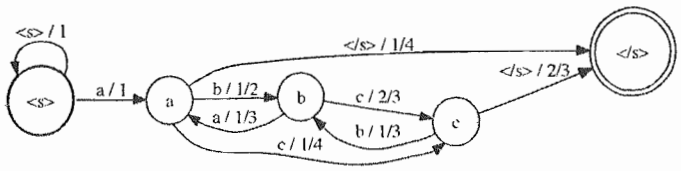
The scores assigned by this model are not normalized and so they do not form correct conditional probability distributions. Hence perplexity and other operations on language models (e.g. entropy-based pruning) are not straightforward. These models can also be very large. The benefits of the model are that the quantities are very easily estimated, especially in a distributed environment with counts gathered from partitioned text collections; this is largely due to the simplistic backoff scheme and

the lack of discounting. This is a 'zero-backoff' model, so that an N-gram probability is generated for all N-grams occurring in the text. The quality of the model can be expected to improve since (as with other N-gram strategies) model components approach the maximum likelihood estimates; hence this model can be expected to perform comparable to other N-gram modeling techniques as the training data size increases, as has been observed in practice. [15%]

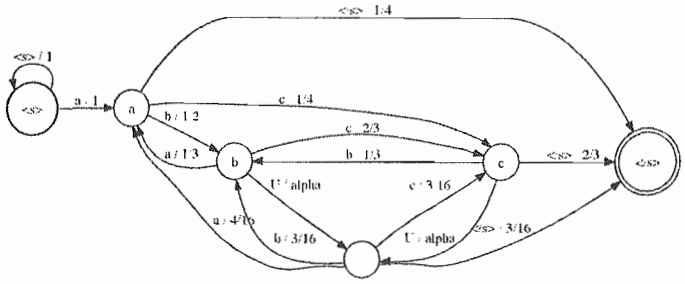
(d)(i) The ML estimate of the bigram language model is derived by dividing the bigram count by the count of the unigram history [10%]

Bigram Probability	Bigram Probability
P(a | <s>) = 3/3	P(a | b) = 1/3
P(b | a) = 2/4	P(c | b) = 2/3
P(c | a) = 1/4	P(b | c) = 1/3
P(</s> | a) = 1/4	P(</s> | c) = 2/3

(d)(ii) [15%]



(d)(iii)



In the picture, the U arcs are failure arcs leading to the backoff state. These arcs have weight α, corresponding to the stupid backoff from bigram to unigram. From the back off state, arcs with the unigram probabilities lead to states consistent consistent with the unigram history. With the failure arcs, paths through the back off state are allowed onl y when a bigram is not present in the model. The score applied to the sequence is exactly that of the stupid backoff bigram. If epsilon arcs are used instead of failure arcs, paths through the backoff state are possible even if the bigram pro bability should be applied. With multiple paths through the lattice, it may happen (depending on the semiring) that the cor rect score is not assigned to an input sequence. [20%]

4. *Machine Translation*

(a) Alignment Error measures the number of non-NULL word alignments by which the automatic word alignment differs from the refer ence word alignment. Suppose there are two sets of alignments, e.g.

- B : automatic word alignments ← *produced by an alignment model*
- B': reference word alignments ← *created by humans*

Alignment error is computed as follows
Step 1. Remove the NULL word links from B' and B to form $\bar{B}'$ and $\bar{B}$
Step 2. Compute $AE(B, B')$:

$$AE(B, B') = \frac{|\bar{B}'| + |\bar{B}'| - 2\,|\bar{B} \cap \bar{B}'|}{|\bar{B}'| + |\bar{B}'|}$$

Alignment Error is often used as an intermediate quality measure in translation system development. If a portion of the parallel text collection is held out from the alignment model training, it can be be manually word-aligned by transcribers f luent in both languages and then used as a reference for computing alignment error. The value of this is that variations in training sets, model configurations, and parameter estimation schemes can be assessed without running a full translation system. Alignment Error has been found to be a good, if not perfect, indicator of the quality of word alignment models: large reductions in alignment error are often correlated with improvements in translation quality. [20%]

(b)

$$P(f_1^J, a_1^J, J|e_0^I) = \frac{1}{I^J}\, p_L(J|I) \prod_{j=1}^{J} p_T(f_j|e_{a_j}) \text{ - Model-1}$$

$$P(f_1^J, a_1^J, J|e_0^I) = p_L(J|I) \prod_{j=1}^{J} p_T(f_j|e_{a_j})\, p_{M2}(a_j|j, I) \text{ - Model -2}$$

[20%]

(c) In a Flat Start Training Procedure, the goal is to gradually increase the complexity of the alignment distribution during training. A typical procedure runs as follows

Model-1

- Model-1 Initialization – set $p_T(f|e)$ to be uniform
- Perform EM (or Viterbi) parameter estimation until some stopping criterion is met
- Output: Generate Model-1 word-aligned parallel text via Viterbi alignment

Model-2

- Model-2 Initialization – find $p_{M2}(i|j,J,I)$ and $p_T(f|e)$ using the word-alignments from Step 1 (c)
- Perform EM (or Viterbi) parameter estimation until some stopping criterion is met
- Output: Generate Model-1 word-aligned parallel text via Viterbi alignment

The stopping criteria are somewhat arbitrary and are set based on experimental performance: the procedure could stop after a fixed number of iterations or when relative likelihood gains drop below a threshold. [20%]

(d)(i)

$$\begin{aligned}
P(f_1^J \mid e_1^I) &= \sum_{a_1^J} P(f_1^J, a_1^J | e_0^I) = \sum_{a_1=0}^{I} \cdots \sum_{a_J=0}^{I} P(f_1^J, a_1^J | e_0^I) \\
&= \sum_{a_1=0}^{I} \cdots \sum_{a_J=0}^{I} \prod_{j=1}^{J} p_{M2}(a_j|j,I,J)\, p_T(f_j|e_{a_j}) \\
&= \sum_{a_1=0}^{I} p_{M2}(a_1|j,I,J)\, p_T(f_1|e_{a_1}) \times \\
&\qquad \sum_{a_2=0}^{I} p_{M2}(a_2|j,I,J)\, p_T(f_2|e_{a_2}) \times \\
&\qquad\qquad \cdots \times \sum_{a_J=0}^{I} p_{M2}(a_J|J,I,J)\, p_T(f_J|e_{a_J}) \\
&= \prod_{j=1}^{J} \sum_{i=0}^{I} p_{M2}(i|j,I,J) p_T(f_j|e_i)
\end{aligned}$$

[20%]

(d)(ii) Using Bayes' Rule

$$P(a_{j'} = i' | f_1^J, e_1^I) = \frac{P(a_{j'} = i', f_1^J | e_1^I)}{P(f_1^J | e_1^I)}$$

Similar to the derivation for (i),

$$P(a_{j'} = i', f_1^J | e_1^I) = p_{M2}(i'|j',I,J) p_T(f_{j'}|e_{i'}) \prod_{j=1, j\neq j'}^{J} \sum_{i=0}^{I} p_{M2}(i|j,I,J) p_T(f_j|e_i)$$

Therefore

$$\begin{aligned}
P(a_{j'} = i' | f_1^J, e_1^I) &= \frac{p_{M2}(i'|j',I,J) p_T(f_{j'}|e_{i'}) \prod_{j=1, j\neq j'}^{J} \sum_{i=0}^{I} p_{M2}(i|j,I,J) p_T(f_j|e_i)}{\prod_{j=1}^{J} \sum_{i=0}^{I} p_{M2}(i|j,I,J) p_T(f_j|e_i)} \\
&= \frac{p_{M2}(i'|j',I,J) p_T(f_{j'}|e_{i'})}{\sum_{i=0}^{I} p_{M2}(i|j',I,J) p_T(f_{j'}|e_i)}
\end{aligned}$$

[20%]