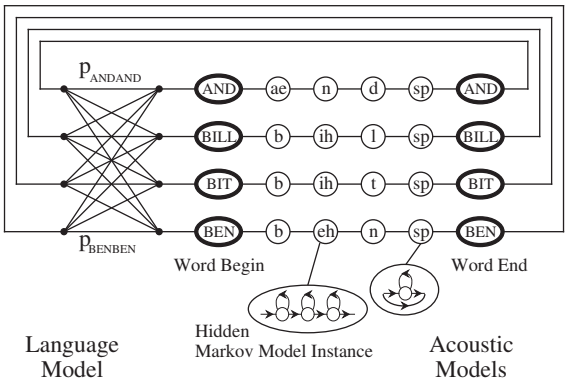
Prof P C Woodland

Solutions: 4F11 Speech and Language Processing, 2014

1. *Decoding and Context-Dependent Phone Models*

   (a) Taking as an example the simple 4 word (AND, BILL, BIT, BEN) from lectures then use: [15%]



   The bigram log probabilities are explicit in the network.

   (b) The Viterbi algorithm can be applied to the above structure (it is an HMM) noting that between models/words the transition probabilities come from the language model and not the internal phone level transition probabilities.

   For an N-state HMM, with observation distributions $b_i(o_t)$ and state transition probabilities $a_{i,j}$ the Viterbi algorithm is as follows.

   Initialization:
   $\phi_1(0) = 1.0$
   $\phi_j(0) = 0.0 \quad \text{for} \quad 1 < j < N \text{ and } \phi_1(t) = 0.0 \quad \text{for} \quad 1 \leq t \leq T$

   Recursion:
   `for` $t = 1, 2, \ldots, T$
   $\ldots$ `for` $j = 2, 3, \ldots, N-1$
   $\ldots\ldots$ compute $\phi_j(t) = \max_{1 \leq k < N} [\phi_k(t-1) a_{kj}] \, b_j(\mathbf{o}_t)$
   $\ldots\ldots$ store the predecessor node: $\text{pred}_k(t)$

   Termination:

$$p(\mathbf{O}, \mathbf{X}^* | \lambda) = \max_{1<k<N} \phi_k(T) a_{kN}$$

   The sequence of models and hence the sequence of words can be found by backtracing along the best state-level path. It is only necessary to store this path at word boundaries (the token-passing implementation includes explicit ways to do this) for efficiency. Note that an explanation based on token-passing is also a valid solution here! [20%]

(c) If a trigram is used then the two-word (rather than single word) context needs to be made explicit in the network. This can be made explicit in the network but requires a huge number of network copies. There are a number of ways that this can be approached to efficiently decode. First is to generate word lattices with a bigram model and then rescore these with a trigram model. This only requires expansion of the network structure that is present in the lattice and is hence much more efficient. The second is to use a Weighted Finite State transducer model to optimise the overall structure which can greatly compress the structure. [Further answers not covered directly in lectures include the use of dynamic generation of the network graph and only the parts of the graph that are within the pruning beam need to be generated]. [15%]

(d)(i) Triphones are models of individual phones that take into account the immediately preceding and following phonetic context. If the context takes account of phonetic context across word boundaries they are cross-word triphones. This models co-articulation effects well and assuming that variability is just due to context then a single Gaussian may be ok (in practice there are other sources of variation). For a context independent model there is no explicit model of co-articulation but a Gaussian mixture of sufficient order can capture the variation. [15%]

(d)(ii) Phonetic decision trees. A tree is grown at the state level, normally for each of three states in left-to-right phone models. A simple binary tree with yes/no questions at each node is used to split contexts into equivalence classes. The questions are typically about groups of phone contexts and are aimed at generalisation through, for instance, phonetic knowledge of similar phones. The trees are automatically grown and at each stage the question chosen is that which gives the largest increase in approximate log likelihood and/or by an occupancy threshold. At the leaf nodes there are groups of contexts which are tied and used to build models. The advantage is that there is no need to back off to shorter contexts as any triphone context will fall into one class or another. Furthermore the technique can be applied to contexts beyond triphones. Disadvantages are that for efficiency a very simple model of log likelihood using a single Gaussian approximation at all levels is used; and that the tree building is greedy - i.e. it makes only locally optimal data splits. [25%]

(d)(iii) The main issues are unseen models (not an issue if using decision tree based tying) and the significantly greater complexity of decoding. Again need multi-pass recognition, WFSTs or dynamic decoding. [10%]

2. *HMM training*

(a) MFCCs are a good, compact model of the local spectrum. The (normalised) log energy is also very useful in making certain distinctions. The reason deltas and delta-deltas are used is due to the HMM assumptions: in particular that the observations are independent given the state that generated it, and hence that the previous and following observations do not affect the likelihood. Since this is not true for speech, (speech has a high degree of continuity) then the local dynamics can be captured by making them explicit in the observation vector. [15%]

(b) The backward probability needs to be defined so that it can be combined with $\alpha_j(t)$. Hence

$$\beta_j(t) = p(\mathbf{o}_{t+1}, \mathbf{o}_{t+2}, \ldots, \mathbf{o}_T \,|x(t) = j, \lambda)$$

It can be computed recursively backwards in time:
Initialisation:

$$\beta_j(T) = a_{jN} \qquad 1 < j \leq N$$

Recursion:
`for` $t = T-1, T-2, \ldots, 2, 1$
$\ldots$`for` $j = N-1, N-2, \ldots, 1$

$$\beta_j(t) = \sum_{k=2}^{N-1} a_{jk} b_k(\mathbf{o}_{t+1}) \beta_k(t+1)$$

[20%]

(c) To find the posterior probability of state occupation given the observation sequence, first note that multiplying the $\alpha_j(t)$ and $\beta_j(t)$ yields

$$p(x(t) = j, \mathbf{O} \,|\lambda) = \alpha_j(t)\beta_j(t)$$

and hence

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

where $p(\mathbf{O}|\lambda)$ can be computed from either the α or β values e,g,

$$p(\mathbf{O}|\lambda) = \sum_{k=2}^{N-1} \alpha_k(T) a_{kN}$$

[10%]

(d) Re-estimation formula for the mean parameter vector:

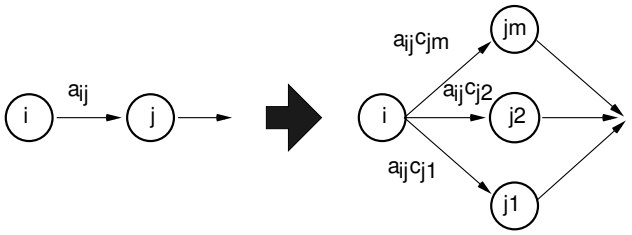
$$\hat{\boldsymbol{\mu}}_j = \frac{\sum_{t=1}^{T} L_j(t) \boldsymbol{o}_t}{\sum_{t=1}^{T} L_j(t)}$$

If there are multiple observation sequences, then sum the numerator statistics and the denominator statistics over the multiple training sequences:

$$\hat{\boldsymbol{\mu}}_j = \frac{\sum_{r=1}^{R}\sum_{t=1}^{T^{(r)}} L_j^{(r)}(t)\mathbf{o}_t^{(r)}}{\sum_{r=1}^{R}\sum_{t=1}^{T^{(r)}} L_j^{(r)}(t)}$$

[15%]

(e)(i) For Gaussian mixture distributions each of the component Gaussians may be considered as a separate state thus:



The posterior probability of a particular Gaussian component of a particular state is:

$$\begin{aligned} L_{jm}(t) &= P(x(t) = jm|\mathbf{O}, \mathcal{M}) \\ &= \frac{1}{p(\mathbf{O}|\mathcal{M})}\sum_{i=2}^{N-1}\alpha_i(t-1)a_{ij}c_{jm}b_{jm}(\mathbf{o}_t)\beta_j(t) \end{aligned}$$

The estimates of the mean will be the similar to the single Gaussian case

$$\hat{\boldsymbol{\mu}}_{jm} = \frac{\sum_{t=1}^{T} L_{jm}(t)\mathbf{o}_t}{\sum_{t=1}^{T} L_{jm}(t)}$$

The component priors can be estimated as

$$\hat{c}_{jm} = \frac{\text{Estimated Number of vectors from comp. } m \text{ state } j}{\text{Estimated number of vectors from state } j}$$

In terms of the posterior probability of state occupation this becomes

$$\hat{c}_{jm} = \frac{\sum_{t=1}^{T} L_{jm}(t)}{\sum_{t=1}^{T} L_j(t)}$$

[25%]

(e)(ii) [Not covered in lectures] The problem is that in maximum likelihood estimation that the likelihood can be maximised in a way that doesn't generalise well by locating Gaussians at training data samples with a variance that tends to zero. This sometimes causes numerical issues with training, but more importantly these Gaussians do not generalise well to test data. The solution is either to modify the estimation procedure to e.g. a MAP-based estimation with a prior on the variance, or to explicitly clamp the variance to a minimum value (which may also be viewed as using a prior distribution). [15%]

3. *Language Models*

(a) N-grams are popular as: they can be computed from real data; they guarantee full coverage of all word sequences; they simultaneously encode syntax, semantics and pragmatics; they concentrate on very local dependencies; they are very simple to compute during recognition - essentially a single table lookup (plus back off computation). [10%]

(b) To find the probability assigned to a word sequence $w_1 \dots w_M$ by an n-gram language model of order N. The sequence is extended to include $w_0 =$ <s> and $w_{M+1} =$ </s>. It follows that

$$\begin{aligned} P(w_0 \dots w_{M+1}) &= \prod_{k=1}^{M+1} P(w_k | w_{k-1} \dots w_{0)} \\ &= \prod_{k=1}^{M+1} P(w_k | w_{k-1} \dots w_{k-N+1)} \end{aligned}$$

where the N-th order Markov assumption is made at the second line. n-gram probabilities of order less than N are used at the start of the sentence, as needed for the first $N-1$ words. [20%]

(c)(i) For a trigram language model, counts $f(w_i, w_j, w_k)$ are collected for all trigrams w_i, w_j, w_k, counts $f(w_i, w_j)$ for all bigrams w_i, w_j and counts $f(w_i)$ for all unigrams w_i in the text corpus (the lower order counts can be found by summing the higher order counts). For an interpolated language model based on a linear combination of trigram, bigram, and unigram counts:

$$\hat{P}(w_k | w_i, w_j) = \lambda_3 \frac{f(w_i, w_j, w_k)}{f(w_i, w_j)} + \lambda_2 \frac{f(w_j, w_k)}{f(w_j)} + \lambda_1 \frac{f(w_k)}{\sum_i f(w_i}$$

where the lambda are positive and $\lambda_1 + \lambda_2 + \lambda_3 = 1$. Note that there must be a scheme to tie the weights across e.g. different count ranges. It would be expected that for high counts that λ_3 might be close to one for example, and for low counts of the trigram the count would be small. Note also need to deal with the situation that the trigram doesn't occur in which case the trigram count may be zero. [20%]

(c) (ii) To set the weights using *deleted interpolation*, need to first divide the text collection nto several parts. One part is held out; the counts $f(\cdot)$ are estimated on the other parts and the trigram $\hat{P}(w_k | w_i, w_j)$ is estimated, as above, and the component models are kept individually. For the heldout data, the training objective

$$P(\text{held out}) = \prod_i \hat{P}(w(i) | w(i-2), w(i-1)$$

The parameters λ are varied to optimise the likelihood of the held-out set. This process is repeated by rotating the held-out set over all the divided parts and hence able to optimise the the interpolation weights (note that as above the weights are not normally trigram specific). [20%]

(c) (iii) For a trigram language model based on the stupid back-off, the stupid back-off assigns scores, rather than correctly normalised probabilities as done by the interpolated language model, to word sequences as

$$S(w_k|w_i,w_j) = \begin{cases} \frac{f(w_i,w_j,w_k)}{f(w_i,w_j)} & \text{if } f(w_i,w_j,w_k) > 0 \\ \alpha\, S(w_k|w_j) & \text{otherwise} \end{cases}$$

where the bigram score $S(w_k|w_j)$ is found by a similar unnormalised backing off to the unigram distribution if $f(w_j,w_k) = 0$.

The stupid backoff is much easier to compute and works particularly well for recognition/translation tasks with very large amounts of training data. However, it cannot be used if correctly normalised probabilities are needed. [20%]

(d) Perplexity (PP) is related to the entropy (H) of the language model

$$\text{PP} = 2^H \quad \text{or} \quad H = \log_2 \text{PP}$$

The probability of a word sequence is often decomposed into the product of word-prediction conditional probabilities:

$$P(w(1)w(2)\dots w(M)) = \prod_{k=1}^{M} P\left(w(k)|w(1)\dots w(k-1)\right)$$

The entropy of this sequence is (letting $M \to \infty$ to obtain a good estimate)

$$H = \lim_{M\to\infty} -\frac{1}{M}\sum_{k=1}^{M} \log_2 P(w(k)|w(1)\dots w(k-1))$$

The perplexity is given by

$$\begin{aligned} PP &= \lim_{M\to\infty} \left(P(w(1)w(2)\dots w(M))^{-\frac{1}{M}}\right) \\ &= \lim_{M\to\infty} \left(\prod_{k=1}^{M} P(w(k)|w(1)\dots w(k-1))\right)^{-\frac{1}{M}} \end{aligned}$$

In practice, will only have S sequences of words on which to estimate the entropy of the language model. The entropy is

$$H = -\frac{1}{\sum_{s\in S} M_s}\left(\sum_{s\in S}\sum_{k=1}^{M_s} \log_2 P(w(k)|w(1)\dots w(k-1))\right)$$

When we compute the perplexity using this over a corpus of test sentences, we get the Test Set Perplexity which is the value normally quoted.

The interpolated LM can be assessed using perplexity; perplexity is directly related to the criterion used to optimise the interpolation weights. However, the stupid backoff scheme does not yield proper log-likelihoods and so perplexity as computed under that model is not valid. [10%]

4. *Machine Translation*

(a) In statistical machine translation, *fertility* is a probability distribution over the number of target words each source word can generate. It approximates phrase modelling. The *distortion model* describes how target language words are distributed throughout the target sentence when generated from a single source word. [15%]

(b) Modelling power and computational tractability of IBM Model-1 and IBM Model-4. IBM 4 is a much more powerful model. It contains a lexical translation model and a NULL translation model, as does Model 1, but model 4 also attempts to capture fertility and distortion. However, Model-4 is deficient, in that it assigns some probability mass to non-sentences so that it is not a proper probability distribution over alignments and translations. Parameter estimation and (Viterbi) alignment are also difficult (impossible) to implement exactly. Model-1 is less detailed, but computationally very efficient. [15%]

(c)(i) For the sentence $e_1^I = e_1 \dots e_I$ and its translation $f_1^J = f_1 \dots f_J$ from the parallel text corpus, the alignment variables a_j take values in $[1, \dots, I]$ so that $a_j = i$ implies that $f_j \leftrightarrow e_i$. This can be extended to allow alignment to NULL. [10%]

(c)(ii) Alignment likelihood $P(f_1^J, a_1^J, J|e_1^I)$ under Model-2.

$$\begin{aligned} P(f_1^J, a_1^J, J|e_1^I) &= P(f_1^J|J, a_1^J, e_1^I)\, P(a_1^J|J, e_1^I)\, P(J|I) \\ &= \textstyle\prod_{j=1}^J p_T(f_j|e_{a_j})\, P_A(a_1^J|J, I)\, P(J|I) \end{aligned}$$

Under Model-2, the alignment distribution has the form

$$P_A(a_1^J|J, I) = \prod_{j=1}^{J} p_{M2}(a_j|j, J, I)$$

[15%]

(c)(iii) To derive an expression for calculation of the probability $P(a_j = i|f_1^J, e_1^I)$ under Model-2. Following the second examples paper,

$$P(a_j = i|f_1^J, e_1^I) = \frac{p_{M2}(i|j, I, J)p_T(f_j|e_i)}{\sum_{i'} p_{M2}(i'|j, I, J)p_T(f_j|e_{i'})}$$

To update the component distributions of the model, the parameters are updated as

$$p_{M2}(i|j, I, J) = \frac{\#_{M2}(i, j, J, I)}{\sum_{i'} \#_{M2}(i', j, J, I)}$$

where $\#_{M2}(i, j, J, I)$ is the accumulation over the parallel text of the $P(a_j = i|f_1^J, e_1^I)$ for sentence pairs of lengths I and J. Similarly,

$$p_T(f|e) = \frac{\#_T(f \leftrightarrow e)}{\sum_{f'} \#_T(f' \leftrightarrow e)}$$

where where $\#_T(i, j, J, I)$ is the accumulation over the parallel text of the $P(a_j = i|f_1^J, e_1^I)$ for sentence pairs of lengths I and J for which $e_i == e$ and $f_j == f$. [25%]

(d) Parallel text consists of French and English travel documents and automatic methods that could be used to create a English-French dictionary for word to word translation. The word-to-word translation distribution $P_T(f|e)$ is a more general case of a translation dictionary. The distribution estimated over the parallel text will be extremely large, and also extremely noisy, and so not directly suitable for presentation to humans. Some scheme for restricting the number of possible translations of each source word will have to be introduced. One approach could be to prune $P_T(f|e)$ to contain only a few of the most likely candidates for each word. Another approach could be to align the parallel text under the model and extract the very frequently occurring words and phrases from the aligned text. These two approaches can be used at the word level. For phrase level, it is difficult to extend Model 2, or other word alignment models, to directly model phrasal alignment. A more straightforward approach is to extract translation phrase pairs directly from the word-aligned parallel text. Here too only the most frequently occurring phrase pairs should be kept, as otherwise there may be too many options to present to readers. [20%]