The Hidden Vector State Language Model

Technical Report CUED/F-INFENG/TR.467

Steve Young (sjy@eng.cam.ac.uk)

Cambridge University Engineering Department
Trumpington Street, Cambridge, CB1 2PZ, England

September, 2003

1 Introduction

The limitations of N-gram language models for use in speech recognition and related applications are well-known and often commented on. Despite this, attempts to significantly improve on them have proved to be elusive [1]. Today it is still true that a 4-gram word model smoothed with a 3- or 4-gram class model is very hard to improve on if word error rate is the only concern [2]. Attempts to model long range dependencies have typically yielded small improvements in perplexity but no significant improvements in word error rate [3].

Attempts to use models which can capture the hierarchical structure of language such as those proposed by Charniak [4] and Chelba [5] typically rely on fully annotated tree-bank data. The resulting models work well when compared to N-gram models trained on comparable amounts of data (typically a few million words), but cannot compete with N-grams trained on a billion words. Of course, if some degree of understanding is required, then the ability of these so-called *structured language models* to deliver surface parse trees might make their use more attractive, but even then it is more likely that a multi-pass approach would be better whereby a conventional robust parser is used to parse the lattice output of a conventional N-gram based recogniser.

Recently in the context of data-driven semantic processing, a form of language model has been proposed which is able to represent hierarchical structure whilst at the same time being trainable on unannotated data using EM [6, 7]. This model is called the Hidden Vector State (HVS) model and it has been shown to be effective in learning to decode semantic classes without the benefit of fully annotated data [8]. This paper discusses the use of the Hidden Vector State Model for language modelling.

2 The Basic HVS Model

The HVS model is a regular HMM in which each state encodes history in a fixed dimension stack-like structure. When used as a language model, the joint probability of a word sequence $W_1^T = w_1, \ldots, w_T$ and the corresponding state sequence $S_1^T = s_1, \ldots, , s_T$ is as for any HMM

$$P(W_1^T, S_1^T) = \left\{ \prod_{t=1}^{T} P(s_t|s_{t-1}) P(w_t|s_t) \right\} P(s_{T+1}|s_T) \tag{1}$$

where it is understood that $s_0 = s_{T+1} = s_\#$, the sentence boundary state. The probability of W_1^T is then found be summing over all state sequences, or more likely performing a Viterbi decode.

The key to the HVS model is that each state corresponds to an internal (hidden) stack where each element of the stack is a label chosen from a finite set of cardinality $M+1$. The interpretation

of these labels is arbitrary but it helps to think of them as class (or category) labels such as might be used to label the nodes of a parse tree. Here they are denoted here as $\mathcal{C} = \{c_1 \ldots c_M, c_\#\}$ where $c_\#$ is understood to correspond to sentence boundaries and by definition

$$s_\# = [c_\#, c_\#, ...] \tag{2}$$

Note that when the state stack is written as a row vector in this way, the most recently pushed element on the stack is always leftmost. Class labels may also be indexed in which case, the most recent stack element is at index position 1 and the oldest is at index position D.

Viewed as a stack, state transitions are decomposed into two sub-steps as illustrated in Figure 1 which highlights the transition from time $t-1$ to t:

1. exactly n_t class labels are popped off the stack
2. exactly one new class label c_t is pushed onto the stack

To keep the stack a constant size, it is convenient to assume that any vacant stack positions resulting from the pop operation are refilled with $c_\#$. When $n = 0$, then the oldest (rightmost) element on the stack is lost following the subsquent push of a new class label.

The HVS model is a restricted form of stochastic push-down automata. Specifically, it is a right-branching stack automata. The number n_t of elements to pop and the choice of the new class label c_t to push are determined probabilistically such that

$$P(s_t|s_{t-1}) = P(n_t|s_{t-1})P(c_t|s^n_{t-1}) \tag{3}$$

where the intermediate state s^n is just s with n class labels popped off and

$$s_t = [c_t, \Phi_1^{D-1}(s^n_{t-1})] \tag{4}$$

The function Φ is a stack access function such that $\Phi_i^j(s)$ denotes the sequence of class labels in positions i to j of stack s and $\Phi_1^D(s) \equiv s$

Notice, that if $n = 0$ always, and if $P(w|s) = P(w|\Phi_1^1(s))$ then this model is essentially a class-based N-gram. When n is allowed to be greater than zero, recent class labels are discarded in favour of retaining older labels. Thus for example, in Figure 1 the class label corresponding to w_1 is retained on the stack until time t. At time $t-1$, the discarded class label c_x might correspond to an embedded phrase. Notice that n_t is conditioned on the identity of c_x but c_t is not. Thus the former distribution can encode embedding, whereas the latter focusses on modelling long range dependencies.

3 Training

Assuming that the state space is fixed, training of the HVS model follows the standard HMM forward-backward re-estimation procedure. The auxiliary function is

$$\sum_S P(S|W,\lambda) \log P(S,W|\hat{\lambda}) = \sum_{C,N} P(C,N|W,\lambda) \log P(W,C,N|\hat{\lambda}) \tag{5}$$

where S, W, C and N are sequences of states, words, class labels and stack pops, respectively. Separating out terms in W, C and N and maximising subject to the usual constraints leads to the re-estimation formulae

$$\hat{P}(w|s) = \frac{\sum_t P(s_t = s|W,\lambda)\delta(w_t = w)}{\sum_t P(s_t = s|W,\lambda)} \tag{6}$$

$$\hat{P}(c|s^n) = \frac{\sum_t P(s_t = [c, \Phi_1^{D-1}(s^n)]|W,\lambda)}{\sum_t P(s^n_{t-1} = s^n|W,\lambda)} \tag{7}$$

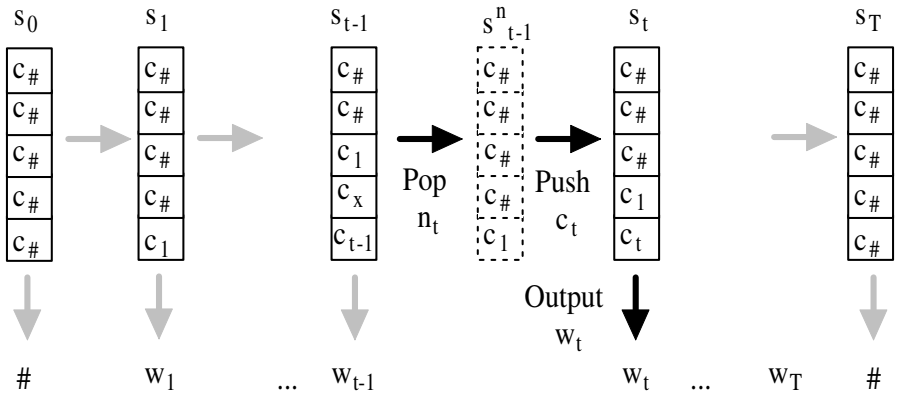

Figure 1: The HVS Language Model

$$\hat{P}(n|s) = \frac{\sum_t P(n_t = n, s_{t-1} = s|W, \lambda)}{\sum_t P(s_{t-1} = s|W, \lambda)} \tag{8}$$

where $\delta(w_t = w) = 1$ iff the word at time t is w.

To calculate the numerator and denominators in equations 6, 7 and 8, define α and β as follows:

$$\alpha_s(t) = P(w_1 \dots w_t, s_t = s|W, \lambda) \tag{9}$$

$$\beta_s(t) = P(w_{t+1} \dots w_T|s_t = s, W, \lambda) \tag{10}$$

then omitting the conditioning W and λ for clarity

$$P(s_t = s) = \alpha_s(t)\beta_s(t) \tag{11}$$

$$P(s^n_{t-1} = s^n) = \sum_{\{s|\Phi_2^D(s)=\Phi_1^{D-1}(s^n)\}} \alpha_s(t)\beta_s(t) \tag{12}$$

$$P(n_t = n, s_{t-1} = s) = \alpha_s(t-1)P(n_t = n|s_{t-1} = s)\sum_c P(c|s^n)P(w_t|s')\beta_{s'}(t) \tag{13}$$

where $s' = [c, \Phi_1^{D-1}(s^n)]$.

The values of α and β are found by the usual recursions

$$\alpha_s(t) = \left\{ \sum_{\{s'|s' \Rightarrow s\}} \alpha_{s'}(t-1)P(s|s') \right\} P(w_t|s) \tag{14}$$

and

$$\beta_s(t) = \sum_{\{s''|s \Rightarrow s''\}} P(s''|s)P(w_{t+1}|s'')\beta_{s''}(t+1) \tag{15}$$

where the notation $\{s'|s' \Rightarrow s\}$ denotes all states s' from which there is a legal transition to s.

4 Data Representation and Smoothing

If the HVS model state space $\mathcal{S}$ is fully populated then

$$|\mathcal{S}| = M^D \tag{16}$$

The class size M will be of the order of 10^2 to 10^3 and D will be at least 3 or 4 to be useful. Thus, the complete state space of the HVS is very large.

In practice therefore, only that part of the state space which is actually observed in the training data is stored, and probabilities over unseen states are computed recursively using back-off [9]. For example, for the word distributions if $P(w|s) = P(w|\Phi_1^D(s))$ has not been seen then it is computed recursively using

$$P(w|\Phi_1^d(s)) = B(\Phi_1^d(s)).P(w|\Phi_1^{d-1}(s)) \tag{17}$$

and similarly for the distributions over n and c.

Let the set of seen words at level d be denoted by $\Omega(d)$, then the back-off weights $B(\Phi_1^d(s))$ are chosen to ensure that the total back-off mass at level $d-1$ is equal to the unassigned probability mass at level d, i.e.

$$B(\Phi_1^d(s)) = \frac{1 - \sum_{w \in \Omega(d)} P(w|\Phi_1^d(s))}{1 - \sum_{w \in \Omega(d)} P(w|\Phi_1^{d-1}(s))} \tag{18}$$

Back-off depends, of course, on there being some unused probability mass available from the estimate of the sparse distribution. In the reestimation scheme described in section 3, this is obtained by discounting the numerator and denominators of the reestimation update formulae 6 to 8. Suitable discount schemes have been extensively covered in the literature [10, 11, 12].

The use of back-off in the HVS model suggests that the three core probability distributions should be stored in a tree-like structure in which each node corresponds to a class label, the immediate descendants of the root node correspond to the class labels in stack element 1 and the leaf nodes correspond to the class labels in stack element D. This is illustrated in Figure 2 which shows a tree for a stack of depth 3 and just two classes. Attached to each node at depth d is a back-off probability $P(w|\Phi_1^d(s))$ for each word w seen at that level, and a back-off weight for the unseen words at that level[1].

5 Initialisation and Tree-Growing

Given some initial distribution for the HVS parameter set, the parameter tree can be grown as required during the parameter update phase, i.e. the basic training cycle is

1. for each training sentence S:
   (a) calculate the α and β "arrays" using equations 14 and 15
   (b) for every state with an occupation count above some threshold, if that state is not in the tree then add it.
   (c) add the (discounted) counts for each seen state s in sentence S to the numerator and denominator accumulators for s
2. when all training sentences have been processed
   (a) update the "leaf probabilities" ie the seen states using equations 8, 7 and 6.
   (b) compute the back-off probabilities attached to the intermediate tree nodes by summing the distributions attached to the immediate ancestors

[1] The back-off weights are not shown explicitly in the figure

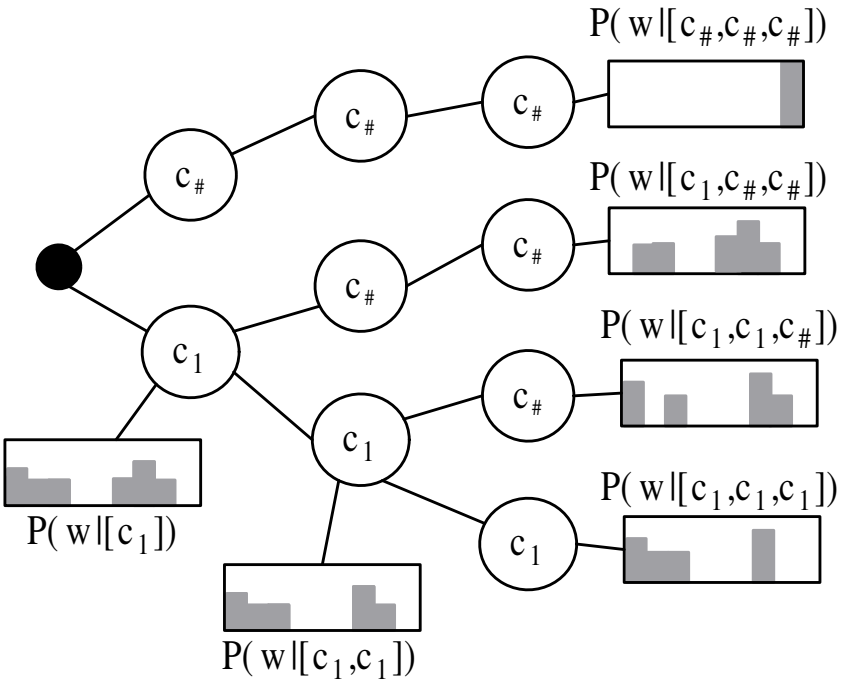

Figure 2: Tree Structure for Storing HVS Probability Distributions and Back-Off Weights

(c) compute the back-off weights at all nodes using equation 18

The obvious initialisation to start the above process off is to use conventional Kneser-Ney bigram clustering to generate a class-based 2-gram with parameters $P(c|c')$ and $P(w|c)$ such that

$$P(n|s) = \delta(n, 0) \tag{19}$$

$$P(w|s) = P(w|\Phi_1^1(s)) = P(w|c) \tag{20}$$

$$P(c|s^n) = P(c|\Phi_1^1(s^n)) = P(c|c') \tag{21}$$

From this initial model, additional states can be introduced by smoothing the distribution on n, gradually giving mass to $P(n = 1|s)$, $P(n = 2|s)$, etc. This will provide more varied stack contexts for the remaining distributions and the "seen state-space" will grow.

6 Conclusions

This paper has described how the HVS model can be used for language modelling. The key difficulty is initialising the model and establishing the underlying state space. Once this is done, training using EM should in principle be straightforward. Work is now progressing to implement the model and explore these issues experimentally.

References

[1] F Jelinek. Up from trigrams. In *Proc Eurospeech*, pages 1037–1040, Genoa, 1991.

[2] T Hain, PC Woodland, TR Niesler, and EWD Whittaker. The 1998 htk system for transcription of conversational telephone speech. In *Proc IEEE Int Conf Acoustics Speech and Signal Processing*, pages 57–60, Phoenix, 1999.

[3] R Rosenfeld. A maximum entropy approach to adaptive statistical language modelling. *Computer Speech and Language*, 10(4):187–228, 1996.

[4] E Charniak. Immediate-head parsing for language models. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, 2001.

[5] C Chelba and F Jelinek. Structured language modeling. *Computer, Speech and Language*, 14(4):283–332, 2000.

[6] SJ Young. The statistical approach to the design of spoken dialogue systems. Technical Report CUED/F-INFENG/TR.433, Cambridge University Engineering Department, 2002.

[7] SJ Young. Talking to machines (statistically speaking). In *Int Conf Spoken Language Processing*, Denver, Colorado, 2002.

[8] Y He and SJ Young. Hidden vector state model for hierarchical semantic parsing. In *Proceedings Int Conf Acoustics Speech and Signal Processing*, Hong Kong, 2003.

[9] SM Katz. Estimation of probabilities from sparse data for the language model component of a speech recogniser. *IEEE Trans ASSP*, 35(3):400–401, 1987.

[10] IJ Good. The population frequencies of species and the estimation of population parameters. *Biometrika*, 40(3):237–264, 1953.

[11] TC Bell, JG Cleary, and IA Witten. *Text Compression*. NJ, 1990.

[12] H Ney, U Essen, and R Kneser. On structuring probabilistic dependences in stochastic language modelling. *Computer Speech and Language*, 8(1):1–38, 1994.