The Hidden Vector State Language Model

Vidura Seneviratne and Steve Young

Cambridge University Engineering Department
Trumpington Street, Cambridge, CB2 1PZ, England.
{vps26,sjy}@eng.cam.ac.uk

Abstract

The Hidden Vector State (HVS) model extends the basic Hidden Markov Model (HMM) by encoding each state as a vector of stack states but with restricted stack operations. The model uses a right branching stack automaton to assign valid stochastic parses to a word sequence from which the language model probability can be estimated. The model is completely data driven and is able to model classes from the data that reflect the hierarchical structures found in natural language. This paper describes the design and the implementation of the HVS language model [1], focusing on the practical issues of initialisation and training using Baum-Welch re-estimation whilst accommodating a large and dynamic state space. Results of experiments conducted using the ATIS corpus [2] show that the HVS language model reduces test set perplexity compared to standard class based language models.

1. Introduction

A language model estimates the probability of a word sequence $P(W_1^T)$, for any word string $W_1^T = w_1, w_2, ..., w_T$. This can be decomposed as a product of conditional probabilities (1) and is for most practical implementations approximated further by defining an equivalence class function ϕ on the history W_1^{i-1} thus limiting the context to the last *n–1*, contiguous words (2) resulting in n-gram models.

$$P(W_1^M) = P(w_1)\prod_{k=2}^{M} P(w_k \mid W_1^{k-1}) \tag{1}$$

$$\approx \prod_{i=1}^{m} P(w_i \mid \phi(W_1^{i-1})) = \prod_{i=1}^{m} P(w_i \mid W_{i-n+1}^{i-1}) \tag{2}$$

N-gram models are widely used and have proved to be extremely difficult to improve upon. However, despite their efficiency and simplicity, n-grams are known to be sub-optimal due to issues of data sparcity, inability to capture long distance dependencies and inability to model the nested structural information found in natural language.

Class-based language models aim to alleviate the problem of data sparcity through the introduction of a further classification function that maps the words to a set of classes, based on a syntactic or semantic definition, like part-of-speech (POS) tags or derived through a data driven clustering method. In general, the latter approach produces better class-based language models [3].

Recent research has aimed at enhancing n-grams with the ability to predict words using longer spans and to make use of the syntactic structure of a sentence in language modelling [4], [5]. These models attempt to incorporate hierarchical information into the language model, typically by using a generative LR parser. This assures a generic context-free approach to language modelling but at a high computation and memory cost. The main limitations of these methods are their dependency on the parser performance and on the availability of a treebank of hand annotated training sentences.

Research in the use of HMMs in language modelling has been based on (syntactic) POS tags [6], and (semantic) concept tags [7] with the hidden variable corresponding to a single tag. The Hierarchical HMM (HHMM) [8] generalises the standard HMM to allow the hidden states to represent stochastic processes themselves, and can be used to model hierarchical structure but it is computationally extremely complex.

The HVS model can be seen as a constrained version of the HHMM in which the hidden state is a vector representing the stack of a push-down automaton, constrained to produce only right-branching parses. It has been successfully used in semantic processing for relatively narrow discourse domains using semi-annotated data [9]. The HVS language model aims to dynamically learn hierarchical clusters and model long-range dependencies among words. This paper presents the design and implementation of the HVS language model and describes experiments which show that it can be trained using Estimation-Maximisation (EM), on unannotated data, and that it has the ability to selectively retain and therefore make better use of context compared to standard class n-grams.

The remainder of the paper is organised as follows: in the following section we describe the HVS language model. Section 3 outlines the issues in building an HVS language model and section 4 gives the details of a preliminary experimental evaluation. Finally, section 5 discusses our conclusions and our ongoing and future work.

2. The HVS Language Model

Any Probabilistic Context Free Grammar (PCFG) formalism can be expressed as a first-order vector state Markov model. The HVS language model further constrains the vector state to be a stack, with state transitions restricted to consist of zero or more pop operations followed by a single push operation. Thus, the basic HVS model is a regular HMM in which each state encodes history in a fixed dimension stack-like structure.

Each state consists of a (hidden) stack where each element of the stack is a label chosen from a finite set of cardinality M+1. These can be thought of as class labels that correspond to nodes of a parse tree denoted as $C = \{c_1, ..., c_M, c_\#\}$ where $c_\#$ corresponds, by definition, to sentence boundaries and the empty stack will have a stack filled with $c_\#$. Thus a D depth HVS model state can be characterized by a vector of dimension D with the most recently pushed element at index position 1 (bottom in Fig 1) and the oldest at the index position D (top in Fig 1). Furthermore, to keep the stack a constant size it is assumed that any vacant positions resulting from the pop operation are refilled with c#.

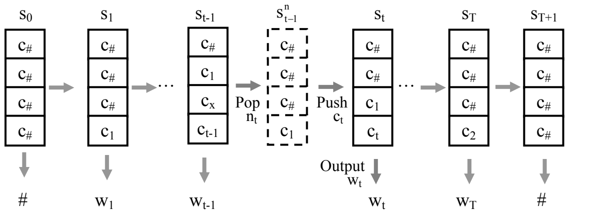

Figure 1: The HVS Language Model

Each HVS model state transition is restricted to the following operations (Figure 1): (i) exactly n_t class labels are popped off the stack, (ii) exactly one new class label c_t is pushed onto the stack. This results in a right branching stack automata. The number of elements to pop (n_t) and the choice of the new class label to push (c_t) are determined probabilistically such that

$$P(\mathbf{s}_t \mid \mathbf{s}_{t-1}) = P(n_t \mid \mathbf{s}_{t-1}) \cdot P(c_t \mid \mathbf{s}^n_{t-1}) \qquad (3)$$

where the intermediate state $\mathbf{s}^n$ is just $\mathbf{s}$ with n class labels popped off and

$$\mathbf{s}_t = [c_t, \Phi_1^{D-1}(\mathbf{s}^n_{t-1})]\,. \qquad (4)$$

The function Φ is a stack access function such that $\Phi_i^j(\mathbf{s})$ denotes the sequence of class labels from i to j of a stack of depth D and $\Phi_1^D(\mathbf{s}) \equiv \mathbf{s}$.

When n is allowed to be greater than zero, recent class labels are discarded in favour of retaining older labels. Notice that n_t is conditioned on all the class labels that are in the stack at t-1 but c_t is conditioned only on the class labels that remain on the stack after the pop operation. Thus the former distribution can encode embedding, whereas the latter focuses on modelling long-range dependencies. Since the stack states are modelled using classes, the data is automatically smoothed and clustered dynamically by the model. Thus an HVS based language model can be seen as a model that both clusters/smoothes the words and builds a dynamic semantic hierarchy among those clusters to predict the next occurring word.

More formally, Let $\mathbf{S}$ and W be a sequence of states and words respectively. Then the joint probability $P(W, \mathbf{S})$ can be expressed as follows

$$P(W,\mathbf{S}) = \prod_{t=1}^{T} P(n_t \mid W_1^{t-1}, \mathbf{S}_1^{t-1}) \cdot P(c_t \mid W_1^{t-1}, \mathbf{S}_1^{t-1}, n_t) \cdot P(w_t \mid W_1^{t-1}, \mathbf{S}_1^{t}) \qquad (5)$$

where

- $\mathbf{S}_1^t$ denotes a sequence of vector states $\mathbf{s}_1 \ldots \mathbf{s}_t$ where each $\mathbf{s}_t$ at word position t is a vector of D class labels
- $W_1^{t-1}, \mathbf{S}_1^{t-1}$ denotes the history up to the word at time t–1
- n_t is the vector stack shift operation which can be up to the number of elements in the stack at time t–1.

The three main components of equation 5 define the pop operation, the push of a new class label and the selection of the most probable word to output. Thus, each hidden vector is defined by three probability distributions; the pop operation probabilities, the class assignment probabilities, and the word output probabilities. By making suitable independence assumptions each of these three components can be approximated as

$$P(W,\mathbf{S}) \approx \prod_{t=1}^{T} P(n_t \mid \mathbf{s}_{t-1}) P(c_t \mid \Phi_1^{D-1}(\mathbf{s}^n_{t-1})) P(w_t \mid \mathbf{s}_t) \qquad (6)$$

The probability of a word sequence $P(W)$ can be estimated by summing over all state sequences (or a Viterbi approximation) of the joint probability in 6.

EM-based parameter estimation can be used to train the model using the auxiliary function

$$\sum_{S} P(\mathbf{S} \mid W, \lambda) \log P(\mathbf{S} \mid W, \hat{\lambda}) = \sum_{C,N} P(N, C \mid W, \lambda) \log P(N, C \mid W, \hat{\lambda}) \qquad (7)$$

By substituting equation 6 in 7 above we can obtain the following re-estimation formulae for each of the components,

$$\hat{P}(n \mid \mathbf{s}) = \frac{\sum_t P(n_t = n, \mathbf{s}_{t-1} = \mathbf{s} \mid W, \lambda)}{\sum_t P(\mathbf{s}_{t-1} = \mathbf{s} \mid W, \lambda)}, \qquad (8)$$

$$\hat{P}(c \mid \mathbf{s}^n) = \frac{\sum_t P(\mathbf{s}_t = [c, \Phi_1^{D-1}(\mathbf{s}^n)] \mid W, \lambda)}{\sum_t P(\mathbf{s}^n_{t-1} = \mathbf{s}^n \mid W, \lambda)}, \qquad (9)$$

and

$$\hat{P}(w \mid \mathbf{s}) = \frac{\sum_t P(\mathbf{s}_t = \mathbf{s} \mid W, \lambda) \delta(w_t = w)}{\sum_t P(\mathbf{s}_t = \mathbf{s} \mid W, \lambda)} \qquad (10)$$

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

The numerators of equations 8, 9, and 10 denote the likelihood of the pop operation, the push operation and the output, respectively, which can be efficiently calculated using the forward backward algorithm defined in terms of allowable stack state transitions.

3. Training the HVS Language Model

The primary issues in the implementation of the HVS language model are the efficient representation of the three core probability distributions (i.e. the three components of equation 5 and 6) that define the model in terms of storage and access, and the efficient calculation of the re-estimation formulae during training.

The state space S, if fully populated, would comprise $|S| = M^D$ states, which is considerable for a reasonable number of classes (e.g. M=100+) and stack depth (e.g. D = 3 to 4). Associated with each of these states are probability vectors representing the pop operation, the push operation and the output, with the dimensions of these vectors corresponding to the stack depth (D), the number of classes

(M) and the vocabulary size (V). Note that the pop operation probability and the output probability are conditioned on a regular state ($\mathbf{s}$), while the push operation is conditioned on an intermediate state ($\mathbf{s}^n$).

Due to data sparcity, in practice, only a part of the state space is observed, and the probabilities that were not observed need to be estimated through backoff. For example, for the word probabilities, if $P(w \mid \mathbf{s}) = P(w \mid \Phi_1^D(\mathbf{s}))$ is not explicitly stored in the model then the backed off probability $P_{BO}(w \mid \Phi_1^D(\mathbf{s}))$ can be computed recursively using

$$P_{BO}(w \mid \Phi_1^D(\mathbf{s})) = B(\Phi_1^{d-1}(\mathbf{s})) \cdot P_{BO}(w \mid \Phi_1^{d-1}(\mathbf{s})) \qquad (11)$$

where the backoff weight $B(\Phi_1^{d-1}(\mathbf{s}))$ is calculated by

$$B(\Phi_1^d(\mathbf{s})) = \frac{1 - \sum_{w \in \Omega(d)} 1 - d(r) \cdot P(w \mid \Phi_1^d(\mathbf{s}))}{1 - \sum_{w \in \Omega(d)} 1 - P_{BO}(w \mid \Phi_1^{d-1}(\mathbf{s}))}\,. \qquad (12)$$

The same method can be used for the other two distributions.

The *d(r)* in 12 above denotes the discount coefficient and was calculated by rounding the fractional counts *a*, and by using a modified version of absolute discounting [10] defined as

$$d(r) = \begin{cases} 1 & a \geq k \\ n_1 \big/ (n_1 + 2n_2) & n_1, n_2 \neq 0, a = 0 \\ \big(a - (n_1 / n_1 + 2n_2)\big) / a & a \neq 0 \end{cases} \qquad (13)$$

Here n_r specifies the number of events that occurred r times and a fixed discounting factor was used if they are zero. Note that the backoff is effectively switched off if the counts exceed the constant *k*. This value controls the amount of backoff in the model and should be small enough to keep the reliable estimates intact while facilitating backoff by discounting some probability from the sparse distribution.

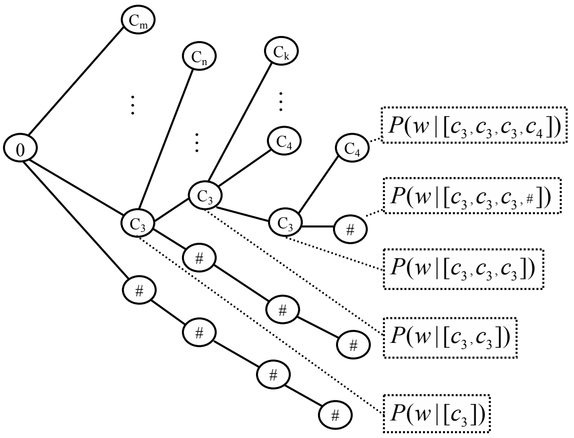
C_m
C_n
C_k
0
C_4
C_4
C_3
C_3
C_3
#
$P(w \mid [c_3, c_3, c_3, c_4])$
$P(w \mid [c_3, c_3, c_3, \#])$
$P(w \mid [c_3, c_3, c_3])$
$P(w \mid [c_3, c_3])$
$P(w \mid [c_3])$

Figure 2: The structure of the class tree

For storage efficiency and to facilitate backoff the three core probability distributions are stored in a tree data structure (Figure 2), called the class tree, in which each node corresponds to a class label with the most recently pushed class label at the first level, and the oldest class label at the leaf level. Then each node corresponds to a state whose stack elements correspond to the class labels along the path from the root to that node. The depth-wise ordering of the class labels within the tree is important for backoff. Each node at depth d is used to store the probabilities for the seen events and the backoff weights for the unseen events at that node. The root node (0) is a special node that is used to store the unigram probabilities.

The initial class tree had only one level of nodes and their parameters were initialised using a uniform distribution for the pop operation and Kneser-Ney class bigram probabilities [10] such that

$$P(n \mid \mathbf{s}) = \mathrm{U}(0,1) \qquad (14)$$

$$P(c \mid \mathbf{s}^n) = P(c \mid \Phi_1^1(\mathbf{s}^n)) = P(c \mid c') \qquad (15)$$

$$P(w \mid \mathbf{s}) = P(w \mid \Phi_1^1(\mathbf{s})) = P(w \mid c) \qquad (16)$$

During each training cycle, the occurrence statistics are accumulated and used to select one or more candidate nodes to be split. A selected node is initially split into the maximum number of child nodes, which is equivalent to the number of classes in the model, and their parameters are initialised to be identical to those of their parent. The backoff is then recalculated and a second pass is performed on this “grown” tree and the statistics obtained are used to both update the parameters and to prune the model by restricting the number of states retained to those that occur more than once. Also, probabilities that are very small are zeroed out thereby retaining only the estimates that the model is confident about.

4. Experiments

A preliminary evaluation of the HVS model was conducted using the ATIS corpus and a conventional class n-gram as a baseline. For each value of n (up to 5) the optimal number of classes was determined by exhaustive search over the test data, thus ensuring that any bias in the choice of class size was in favour of the baseline. We then built HVS models with the same number of classes and iteratively trained them using the procedure described in section 3.

All of the data was used for the training sentences except for the ATIS-3 Nov 93 set, which was used as the developmental set to optimise parameters and the ATIS-3 Dec 94 set, which was used as the test set. The training data had 276K words in 23K sentences while both the developmental set and the test set consisted of 10K words in 1K sentences. The vocabulary size was 1644 and the average sentence length was 10. All hesitations and interjections that had been marked up in the data were removed. All sentences that contained out-of-vocabulary words were removed from the test set.

Two separate classes were assigned for the start and end words and all other words were assigned among the remaining classes. The k, in equation 13, was set to 850.

To study the variation of the perplexity improvement with the depth of the stack, several smaller models were made and the corresponding class n-gram perplexities are given for comparison. It should be noted that a HVS model of depth d corresponds to a class (d+1)-gram. The perplexity values for the model with 52 classes is presented in Table 1.

# of classes in history	2 (3-gram)	3 (4-gram)	4 (5-gram)
52 class n-gram	26.3	25.0	24.9
HVS_52	21.7	20.4	20.1

Table 1: Perplexity for models of varied stack depths trained for 250 iterations.

The baseline class n-gram models were built using the HTK toolkit [11]. The optimum number of classes (yielding the lowest perplexity) was found to be 116 for all the n-gram models. The perplexity values corresponding to each class n-gram model and that of an HVS model with stack depth three is given in Table 2 below.

Data Set	Class 3-gram	Class 4-gram	Class 5-gram	HVS 3d
ATIS-3 Nov93	18.6	18.0	**18.0**	**16.5**
ATIS-3 Dec94	17.3	**16.4**	16.9	**15.5**

Table 2: Perplexity results for models with the optimum number of classes (116).

In all of the above cases the class perplexity values of the HVS models are lower than that of the corresponding class n-gram perplexity. It can be seen from Table 1 that the perplexity values improve as the depth of the stack is increased. To observe the efficiency of the training regime we checked the perplexity at the end of each iteration and observed that even with the current rather naïve node splitting method the training progressed smoothly (Figure 3).

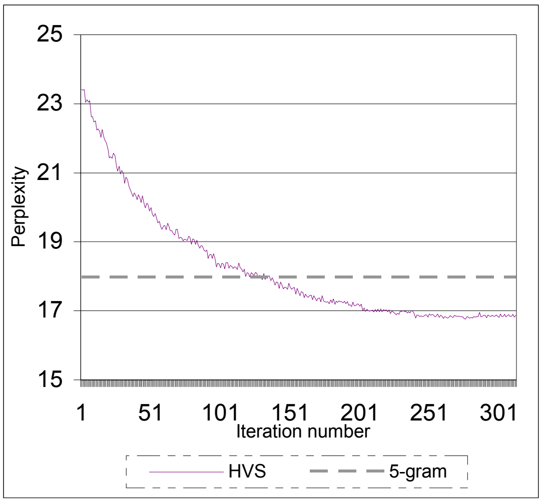

Figure 3: Variation of the perplexity during training using the ATIS-3 Nov 93 set

5. Conclusions and Future Work

This paper has presented the Hidden Vector State model for language modelling and a preliminary experimental evaluation using the ATIS corpus. The perplexity results obtained so far suggest that the HVS language model is able to make better use of context than the standard class n-gram models. Furthermore, inspection of the stack contents during typical parses suggests that the model is tracking to some extent the phrase structure of the data. Thus, the model might be used not only in recognition but also in assisting with meta-data extraction such as punctuation and phrase boundary detection [12].

The fact that the HVS model is trainable using EM enables, in principle at least, the advantages of a structured language model to be extended into the domain of very large training sets where the provision of annotated treebank data is impractical. However, to achieve this in practice, significantly more efficient training algorithms will need to be developed. The specific areas needing development are the backoff computations, the pruning of irrelevant states, and the node splitting strategy, and these will be the focus of future work.

6. References

[1] Young, S. J., "The Hidden Vector State language model", Tech. Report CUED/F-INFENG/TR.467, Cambridge University Engineering Department, 2003.

[2] Price, P., "Evaluation of spoken language systems: the ATIS domain", In *Proc. ARPA HLT Workshop*, 91-95, 1990.

[3] Niesler, T.R., Whittaker, E.W.D., and Woodland, P.C., "Comparisons of Part-of-Speech and automatically derived category-based language models for speech recognition", In *Proc. of the ICASSP*, Washington, 1998.

[4] Chelba, C. and Jelinek F., "Structured language modelling", *Comp. Speech & Language*, 14(4): 283–332, 2000.

[5] Roark, B., "Probabilistic top-down parsing and language modelling", In *Comp. Linguistics* 27(2): 249-276, 2001.

[6] Jelinek**,** F. "Robust Part-Of-Speech tagging using a Hidden Markov Model", IBM Tech. Report, 1985.

[7] Levin, E. and Pieraccini R., "CHRONUS, the next generation", In *Proc. of the 1995 DARPA Speech and Natural Language Workshop*, Texas, 269-271, 1995.

[8] Fine, S., Singer Y., and Tishby N., "The Hierarchical Hidden Markov Model: Analysis and applications", *Machine Learning* 32(1): 41-62, 1998.

[9] He, Y. and Young S.J., "Hidden Vector State Model for hierarchical semantic parsing", In *Proc. of the ICASSP*, Hong Kong, 2003.

[10] Ney, H., Essen, U., and Kneser R., "On structuring probabilistic dependences in stochastic language modeling", *Comp. Speech and Language*, 8:1-38, 1994.

[11] Young, S. J., Kershaw D., Moore G., Odell J., Ollason D., Valtchev V., and Woodland P. C., *The HTK Book*, Cambridge University Engineering Department, URL: http://htk.eng.cam.ac.uk/. 2002.

[12] Liu Y., Shriberg E., Stolcke A., Peskin B., Ang J., Hillard D., Ostendorf M., Tomalin M., Woodland P.C., and Harper M., "Structural metadata research in the EARS program", In *Proc. of the ICASSP*, Philadelphia, 2005.