IDIAP COMMUNICATION

TODE : A Decoder for Continuous Speech Recognition

Darren C. Moore [a]

IDIAP–Com 02-09

December 2002

Dalle Molle Institute for Perceptual Artificial Intelligence • P.O.Box 592 • Martigny • Valais • Switzerland

phone +41 − 27 − 721 77 11
fax +41 − 27 − 721 77 12
e-mail secretariat@idiap.ch
internet http://www.idiap.ch

a darren.moore@idiap.ch

1 Overview

This document describes a new continuous speech decoder, TODE, which is compatible with the Torch machine learning software library. A brief theory of speech recognition is presented followed by a detailed description of the architecture of TODE and the components used in its implementation. Results are presented from experiments that assessed the performance of TODE and compared it with two other commonly-used decoders.

2 Introduction

TODE (TOrch DEcoder) is a continuous speech recogniser based on a time-synchronous beam-search algorithm that is compatible with the Torch machine learning library [1]. It's purpose is to satisfy the general speech decoding needs of researchers at IDIAP and in the wider speech community. TODE has been designed to be a flexible recogniser with a straightforward implementation, that overcomes some of the limitations of other popular decoders while maintaining an acceptable level of efficiency.

This document begins with a brief overview of the theory of decoding. The architecture and implementation of TODE is then discussed in detail. Experimental results comparing the performance of TODE to that of other decoders is then presented, followed by a discussion of future directions.

3 Speech Decoding Theory

The purpose of a speech recogniser is to find the most likely word sequence, $W_{\text{opt}} = w_1 \ldots w_m \ldots w_M$, given a sequence of acoustic observations, $X = x_1 \ldots x_t \ldots x_T$. That is,

$$W_{\text{opt}} = \arg\max_W P(W|X) \tag{1}$$

The solution to the above equation cannot be computed directly, so Bayes theorem of conditional probabilities is applied to give,

$$W_{\text{opt}} = \arg\max_W \frac{P(X|W)P(W)}{P(X)} \tag{2}$$

The denominator in (2) does not affect the choice of word sequence, W, and can therefore be ignored during decoding. Thus the solution requires two probability distributions, the *acoustic model*, $P(X|W)$, and the *language model*, $P(W)$, along with an appropriate search strategy.

3.1 Acoustic Model

Typically, hidden Markov models (HMMs) are used to model the production of a sequence of acoustic observations corresponding to a particular utterance. The HMM for a complete utterance is constructed as a concatenation of word-level HMM's.

Using such a HMM framework, the production of a sequence of acoustic observations given a particular string of words depends on the path, $S = s_1 \ldots s_t \ldots s_T$ that is taken through the states of the HMM. The acoustic likelihood, $P(X|W)$, can then be written as

$$P(X|W) = \sum_S P(X, S|W) \tag{3}$$

Instead of summing over all possible sequences of states, the acoustic likelihood is often approximated by considering only the most probable state path (the *Viterbi approximation*) [2] . Using the approximated acoustic likelihood, Equation 2 can be reformulated as,

$$W_{\text{opt}} = \arg\max_{W}\{P(W) \cdot \max_{S} P(X, S|W)\} \tag{4}$$

3.2 Language Model

The language model needs to estimate the (prior) probability of a given sequence of words, $P(w_1 \ldots w_m \ldots w_M)$. This probability is independent of the acoustic observations and serves to restrict the ways in which word models are concatenated to model entire utterances.

The number of possible word sequences grows exponentially with the length of the word sequence, which makes the full computation of P(W) a formidable task even for small values of M. Therefore, the language model probabilities are usually approximated using *N-gram* models, typically with $N = 2$ (bigram model) or $N = 3$ (trigram model):

$$P(w_m|w_1 \ldots w_{m-1}) \quad \cong \quad p(w_m|w_{m-N+1} \ldots w_{m-1}) \tag{5}$$

3.3 Search

The search space associated with (4) can be viewed as a massive network consisting of the states of all word-level HMM's, with a transition from the final state of each word model to the initial states of all word models.

The problem is to take a sequence of acoustic observations and the network of HMM states, and then search for the most likely path through the network. From Equation 4, the search must be performed at both state and word levels. Thus, the goal is to associate the acoustic observation at each time step with the state-word pair that results in the globally most probable path.

A well-known algorithm for performing this search is the *Viterbi* or *Dynamic Programming* (DP) algorithm [3, 4, 2]. At each time frame, (ie. for each acoustic observation), the algorithm examines every state in the network, and hypothesises whether that state forms part of the most likely state path. The algorithm is based on a recursion that exploits the Viterbi approximation in (4) and Bellman's Principle of Optimality (ie. that the global best path includes the best path to an intermediate state). At each time step, the number of paths considered by the search is limited by recombining hypotheses at both the state and word levels.

Each word-level HMM is assumed to consist of a number of emitting states as well as an initial and final state that are both non-emitting. The notation $(s;w)$ is used to denote any state, s, of the HMM associated with word w. $(s = 0;w)$ and $(S;w)$ denote the initial and final states respectively. If the assumption is made that a transition into an emitting state $(s';w)$ occurred at time $t-1$, then at time t, $(s';w)$ will emit the acoustic observation x_t with a certain probability, and a transition to a state, s will occur according to the topology of the HMM.

Using the above notation and assumptions, the DP recurrence equation for word-interior states is,

$$Q(t,(s;w)) = \max_{s'}\{p(x_t,(s;w)|(s';w)) \cdot Q(t-1,(s';w))\} \qquad (s' \neq 0) \tag{6}$$

where $Q(t-1,(s';w))$ an accumulated 'score' at the state, $(s';w)$, at time $t-1$. $p(x_t,(s;w)|(s';w))$ is the joint probability of emitting the acoustic observation, x_t, from state, $(s';w)$, and then making a transition to the state $(s;w)$.

The DP equation for transitions between words is,

$$Q(t,(s = 0;w)) = \max_{v}\{p(w|v) \cdot Q(t,(S;v))\} \tag{7}$$

where $p(w|v)$ is the language model probability of the word, w, following the word, v. Note that the time index, t, is the same on both sides of the equation because transitions between non-emitting states in a HMM occur instantaneously without producing an acoustic observation. The transition from the initial state to successor emitting states within a word model occurs in a similar manner,

$$Q(t,(s;w)) = \max(\ \ Q(t,(s;w))\ \ ,\ \ p((s;w)|(s=0;w))\cdot Q(t,(s=0;w))\ \ ) \tag{8}$$

where $p((s;w)|(s=0;w))$ is the probability associated with the transition from the initial state to an emitting state in the same word model. It is assumed that the calculation of (6), (7) and (8) at each time step occurs in the order presented.

The basic DP algorithm that uses (6), (7) and (8) is

- for each time step, t
  - for all $(s;w)$ (except initial states)
    - calculate $Q(t,(s;w))$ using (6)
  - for all initial states, $(s=0;w)$
    - calculate $Q(t,(s=0;w))$ using (7)
    - for all successors of $(s=0;w)$
      - evaluate (8) and update if necessary

The algorithm propagates hypotheses through the state network. Information about the path each hypothesis takes through the state network is stored with the hypothesis, so that after the DP algorithm has processed all input acoustic observations, the most probable word sequence (or state sequence) can be recovered. This word sequence corresponds to the W_{opt} in (4).

4 Other Decoding Techniques

4.1 Beam Search Decoding

Beam search decoding is a variant of the DP algorithm that exploits its time synchronicity (ie. that the hypotheses in all states are updated in parallel at each time instant). This allows the hypothesis score at a particular state to be directly compared with the hypothesis scores in all other states at a given time point in time [5]. In beam search decoding, a constant *beam-width* is specified and a pruning threshold is calculated at each time step by subtracting the constant from the most promising hypothesis score. Any hypotheses with scores outside of the *beam* are deactivated and removed from further consideration.

With reasonable beam widths, this technique results in considerable computational savings with a comparatively small reduction in recognition accuracy.

The HTK decoder, HVite [6], uses a beam search decoding approach.

4.2 Stack Decoding

The dynamic programming and beam search techniques are time-synchronous, because active hypotheses are extended in parallel at each time instant, and comparisons are only made between hypotheses of the same age. This constitutes a breadth-first search of the state space.

Stack (or A^*) decoding techniques are time-asynchronous, depth-first methods in which the best hypothesis (regardless of age) is selected and extended until it becomes unpromising. The extension is then applied to next best hypothesis, and so on until a complete hypothesis spanning the entire utterance is obtained [7, 8].

Hypotheses with different ages are compared by estimating their total likelihood (ie. the score obtained by extending a partial hypothesis until it spans the entire utterance). The hypothesis with the best total likelihood estimate is selected for extension. For a hypothesis, h, at time, t, the total likelihood estimate is given by,

$$f_h(t) = a_h(t) + b_h^*(t) \tag{9}$$

where $a_h(t)$ is the score of the hypothesis at time, t, and $b_h^*(t)$ estimates the best possible score obtained by extending the hypothesis over the remainder of the utterance. If $b_h^*(t)$ is an upper bound on the exact likelihood, $b_h(t)$, then the stack decoding process is admissible (ie. is guaranteed to find the optimal complete hypothesis) [9].

A priority stack data structure is used during decoding to keep track of the most promising hypotheses. The basic operation of the stack decoder is [7]:

1. initialise the stack with a null hypothesis
2. pop the hypothesis with the best total likelihood estimate from the stack
3. if hypothesis is not at end-of-utterance
   (a) perform acoustic & LM fast matches to obtain a short list of possible extensions
   (b) for each word in the short-list
      i. perform acoustic and language model detailed matches to compute the total likelihood estimate for the new hypothesis
      ii. push the new hypothesis onto the stack
4. if hypothesis is at end-of-utterance, output recognised sentence and terminate
5. go to 2

The Noway decoder [8] uses a modified stack decoding algorithm and is designed to work within a hybrid ANN/HMM framework.

5 TODE Architecture

5.1 Overview

The speech recognition theory presented in Section 3 implies the general system architecture in Figure 1. Note that the acoustic model has been divided into two parts, a *sub-word models* component and a *pronunciation lexicon* component. In practical systems, it is infeasible to train separate HMM's for each word, so words are typically decomposed into sub-word units (eg. phonemes) and models are trained for each unique sub-word unit. The pronunciation lexicon then defines how sub-word models are combined to form complete word models.

The high-level architecture of TODE is based on Figure 1 and the main components (ie. C++ objects) used in its implementation are included in Figure 2. The following subsections describe each component.

The Beam Search decoding technique was chosen for the initial decoder implementation because it is well-suited to both small and medium vocabulary recognition tasks, and functions reasonably well for large vocabulary tasks.

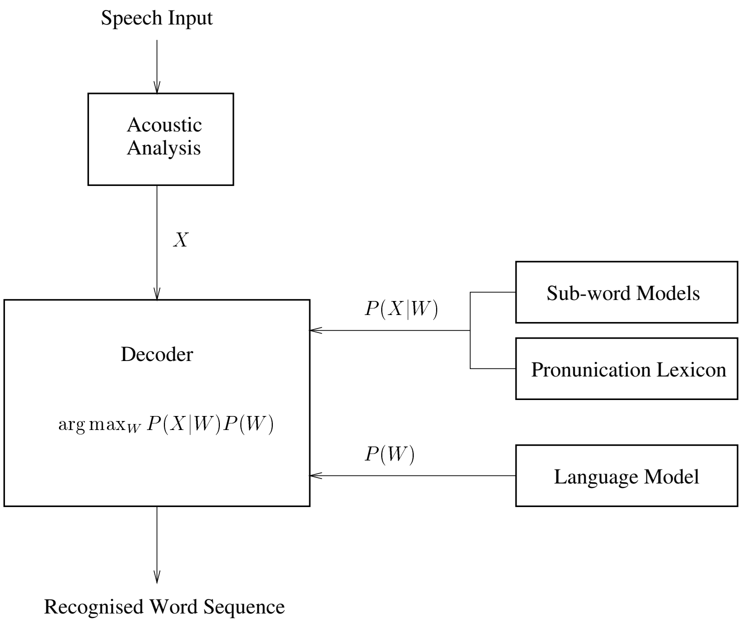

Figure 1: General speech recogniser architecture

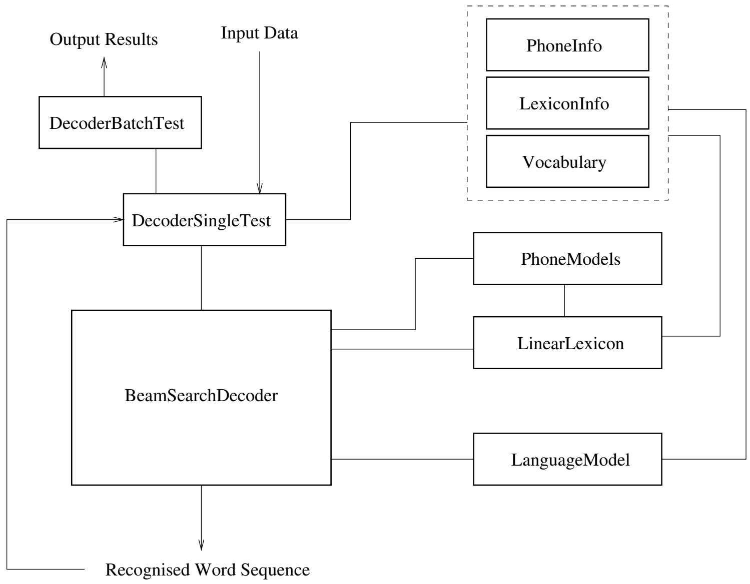

Figure 2: TODE high level architecture

5.2 PhoneInfo

The PhoneInfo object is a simple object containing a list of the names of the phonemes used within the recogniser, as well as indices into the list for 'special' phonemes such as silence or pause. Other objects that access phonemes (eg. LexiconInfo) store indices into this list.

The PhoneInfo object can be initialised using either a HTK format model definition file, a Noway format model definition file, or a file containing a straight list of phoneme names.

5.3 PhoneModels

The PhoneModels object contains all information about the (previously trained) sub-word models that are used during decoding. The ordering of the list of models matches the order of the list of phoneme names in the PhoneInfo object. The information stored for each model includes the HMM topology, an emission probability distribution and a prior probability for the phoneme.

During decoding, the input vector at each time frame is passed to the PhoneModels instance. The PhoneModels instance knows whether the input vector consists of features, or pre-computed emission probabilities. Other objects can request the PhoneModels instance to return the emission probability associated with a particular state in a sub-word model. If the input vectors are features, the PhoneModels instance calculates the requested emission probability and then remembers the value, to make repeated requests for the same emission probability efficient. If the input vectors are emission probabilities, the PhoneModels instance knows the mapping between emitting states of the sub-word models and elements of each input vector.

The DecodingHMM object is used to store all parameters and other data associated with each sub-word HMM. Information about each state is stored in a DecodingHMMState structure, which contains lists of successor states and associated transition probabilities. The DecodingHMMState structure also contains a pointer to a generic *Distribution* object. The Distribution class is implemented within Torch as a base class from which classes that implement specific distributions (GMM's, ANN's, etc) are derived. The Distribution pointer within the DecodingHMMState structure allows the decoding to be independent of the type of emission probability distribution used within the states of the sub-word HMM's.

A SpeechMLP class has been implemented (using many classes from Torch) in order to support hybrid ANN-HMM speech recognition systems. This class is initialised using MLP weights read from a file. It then accepts input feature vectors and calculates the corresponding MLP outputs, which are then used as the emission probabilities for the states in the sub-word HMM's. The SpeechMLPDistr (derived from Distribution) provides the interface between the SpeechMLP and the Distribution pointer in the DecodingHMMState structure.

The PhoneModels object can be initialised using either a HTK format model definition file or a Noway format model definition file.

5.4 Vocabulary

The Vocabulary object contains a list of the names of all unique words that the system can recognise, as well as a record of 'special' words, such as sentence-markers or silence. Other objects (eg. LanguageModel) use indices into the list of words during decoding. Methods are provided to map between word strings and their indices in the list of names (eg. for displaying the result of decoding to the user).

The Vocabulary object can be initialised using a Noway-format dictionary file.

5.5 LexiconInfo

The LexiconInfo object contains information about the pronunciations that can be recognised by the system. The particular sub-word units that comprise each pronunciation are stored as lists of indices into the PhoneInfo object. The Vocabulary entry associated with each pronunciation along with a prior probability are also stored. The LexiconInfo object also stores pointers to the PhoneInfo and Vocabulary objects, and implements a mechanism for mapping unique Vocabulary entries to one or more pronunciations.

The LexiconInfo object can be initialised using a Noway-format dictionary file.

5.6 LinearLexicon

The LinearLexicon object consists of a list of pronunciation-level models that are each constructed by concatenating sub-word models from the PhoneModels object according to the pronunciation information stored in the LexiconInfo object. The ordering of the list of models corresponds to the ordering of the pronunciation entries in the LexiconInfo object. The emitting states of each pronunciation model are mapped back to the correct emitting states in the PhoneModels sub-word models, so that emission probabilities can be efficiently accessed and computed. The prior probability associated with each pronunciation entry in the LexiconInfo object is applied to all transitions out of the initial state of the corresponding pronunciation model.

The states of all pronunciation models forms the state-space that is searched using the DP approach described above.

5.7 LanguageModel

The LanguageModel object implements a tree-like data structure for efficient storage and lookup of probabilities for arbitrary N-gram language models, with full back-off capability. The data structures that have been implemented offer the following advantages:

- A search for an N-gram probability (ie. $P(w_4|w_1, w_2, w_3)$ for $N = 4$) entails N binary searches (if the N-gram exists).
- All lesser order LM probabilities as well as weights required for back-off calculations are retrieved as a side effect of the search for the main N-gram entry with negligible computational overhead. If there is no N-gram entry for a particular sequence of words, then the back-off calculations can be performed immediately without the need for further searches through the tree.

A rudimentary caching scheme has been implemented within the LanguageModel class. A small list of the most recently accessed language model probabilities is maintained, and the first stage of a language model query entails a linear search of the cache. If the desired probability is not in the cache, then the tree structure is traversed and any required back-off calculations are performed.

The LanguageModel data structures are created by reading entries from an ARPA-format language model file.

5.8 DecoderSingleTest

The DecoderSingleTest class manages the decoding of a single utterance. It performs the following functions:

- Reads emission probability or feature input vectors (in a variety of formats) from a specified single-utterance input file, or from an offset into an multi-utterance archive file.
- Packages the input vectors into a format ready to be passed to the decoder
- Invokes the decoder and records the resulting outputs (recognised words, segmentation information, etc)
- Post-processes the decoder output (eg. transform indices into word strings, remove start and end silence words)

- Output the decoding result in one of a variety of formats to a designated output file.
- Measures the CPU time used to decode the utterance.

5.9 DecoderBatchTest

The DecoderBatchTest class manages the decoding of a batch of utterances. A DecoderSingleTest instance is configured for each utterance and after all DecoderSingleTest instances have finished decoding, the results from all utterances are analysed to determine the total number of substitutions, insertions and deletions as well as the overall word recognition rate.

5.10 BeamSearchDecoder

The BeamSearchDecoder class implements the beam search decoding technique described in Section 4. It accepts input feature or emission probability vectors, uses the data stored in the PhoneModels, Vocabulary, LinearLexicon, and LanguageModel objects, and then returns the recognised word sequence along with word-level segmentation information.

At each time frame, the interior state hypothesis with the best (log) score is remembered. A threshold is then calculated by subtracting a pruning constant (specified as an input parameter) from the best score. Any interior state hypothesis with a score below this threshold is deactivated, and removed from further consideration. The same hypothesis pruning technique process is performed on hypotheses in word-end states.

The application of language model probabilities is delayed (ie. does not occur when a hypothesis makes a transition between two words). Instead, language model probabilities are applied to hypotheses when they reach the final state of a word model, which serves to *tune* the hypothesis to reflect the language model constraints. The delayed application of LM probabilities combined with hypothesis pruning significantly reduces the number of language model searches that are required. This is because hypotheses in word models that do not reflect the acoustic observations get pruned before they can be propagated to a final state.

The transition between words is performed according to the standard DP approach. The word-end state hypothesis with the best score is selected, and this becomes the predecessor state for the initial states of all words.

6 Results

6.1 TODE vs. HVite

The performance of TODE was compared to that of the HTK decoder, HVite, for the Numbers corpus recognition task. A set of 80 triphone HMM's were trained using Torch, and were saved in HTK model definition format. Each triphone HMM consisted of 3 emitting states with 10 mixtures per state. 39-element feature vectors were used, comprising 13 MFCCs (including the 0th cepstral coefficient) with their first and second order derivatives. The test set used for the evaluations consisted of 1206 utterances.

All evaluations were performed using a Pentium 4 2GHz PC running Linux.

The purpose of the first experiment was to compare the performance of the two decoders for the simplest recognition task (ie. no language model, no pruning, no tuning of decoder parameters). The results are displayed in Table 1, and verify that the TODE implementation of the dynamic programming algorithm is correct, and executes faster than HVite.

The second experiment compared the performance of the two decoders for the same recognition task when using a back-off bigram language model. The results are displayed in Table 2. The slight difference in WER can be attributed to the delayed application of language model probabilities in

	Execution Time (s)	WER (%)	Ins	Del	Sub
HVite	351.36	6.23	96	43	152
TODE	330.18	6.23	96	43	152

Table 1: TODE vs. HVite results for straight recognition on Numbers.

	Execution Time (s)	WER (%)	Ins	Del	Sub
HVite	361.68	6.19	90	45	154
TODE	320.31	6.17	89	45	154

Table 2: TODE vs. HVite Numbers recognition results using back-off bigram language model.

TODE, as opposed to the "correct" application of bigram LM probabilities by HVite. As with the simple decoding experiments, TODE performs this recognition task faster than HVite.

No further comparative experiments were performed, because HTK does not support language models of higher order than 2.

6.2 Effect of Pruning

The effect of varying the degree of hypothesis pruning within the TODE decoder was assessed. The parameter used to control the amount of pruning is a constant positive log value that gets subtracted from the most promising hypothesis score at each time step to give a pruning threshold. All hypotheses with scores below the threshold are deactivated and removed from further consideration. A small constant results in a narrow beam with all but the few most likely hypotheses being pruned at each time step, and a large constant reduces the number of hypotheses that are pruned. The no-pruning case corresponds to a pruning constant of infinity.

The two graphs in Figure 3 illustrate the effects of varying the pruning constant within TODE for the simple Numbers recognition task described in Section 6.1. The upper plot displays the word recognition accuracy as the pruning constant is increased. The lower plot shows the corresponding execution times.

From Figure 3 it can be seen that applying a pruning constant of 80 has had negligible effect on the recognition accuracy, and has decreased the execution time by a factor of 8 compared with the no-pruning case (see Table 1). Decreasing the pruning constant below 80 results in faster execution times, but has a increasingly negative effect on recognition accuracy.

6.3 TODE vs. Noway

The performance of TODE was compared to that of the Noway decoder for a large vocabulary decoding task. The input to both decoders was emission probability vectors obtained from an ANN trained on the BREF (french) corpus. Each emission probability vector contained 36 elements, representing posterior probability estimates at each time step for each of the 36 phonemes used in the system. A full back-off trigram language model containing approximately 1.2 million bigram entries and 1.8 million trigram entries was used. The test set used for the evaluations consisted of 242 utterances.

The many tuning parameters available within Noway (pruning, probability scaling, etc) were set to values as per the example in the Noway man page. A moderate amount of hypothesis pruning was employed within TODE. The results are displayed in Table 3.

The word error rate is high for both decoders because no effort has been made to tune parameters such as word insertion penalties, language model probability scaling factors and acoustic probability scaling factors in order to achieve optimal performance. It is noted, however, that when optimal

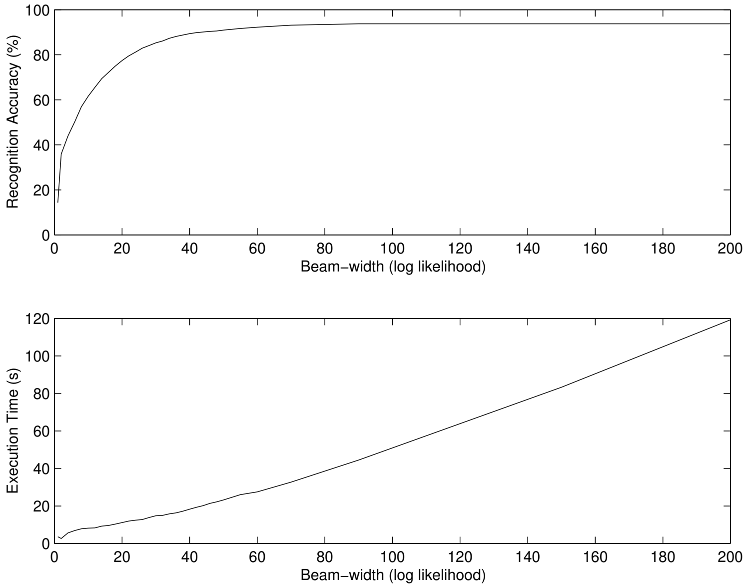

Figure 3: Effect of hypothesis pruning on word recognition accuracy and execution time

	Execution Time	WER (%)	Ins	Del	Sub
Noway	473m 27s	72.7	991	89	1381
TODE	295m 59s	80.7	526	237	1970

Table 3: TODE vs. Noway BREF recognition results.

parameters are used, Noway is by far the better decoder for large vocabulary recognition tasks within a hybrid HMM/ANN framework.

The linear lexicon used by TODE severely limits the execution speed for large vocabulary tasks. This is because the best word-end hypothesis has to be extended to the initial states of *all* pronunciation models at each time step, regardless of the amount of pruning employed. A much more efficient lexicon structure is a tree lexicon [10], where the arcs of the tree represent phonemes and the sequence of arcs from the root to each leaf represents a valid pronunciation. This structure allows pronunciations with common starting phonemes to be compactly represented in the lexicon, and results in a significant improvement in computational efficiency [2].

7 TODE Feature Summary

The major features of TODE are :

- Efficient beam search decoder.
- Can be used with both ANN and GMM-based systems.
- Accepts features or emission probabilities as input.
- Arbitrary N-gram language modelling with full back-off and caching.
- Supports many commonly used file formats (model definition, ANN weights, features, language model, etc).
- Uses a linear lexicon
- Implementation is straightforward, and can be readily modified/upgraded to meet the needs of researchers.
- Easily adapted for use in non-speech decoding applications.
- Fully supported with development ongoing.

8 Future Work

Future work on TODE will focus largely on improving large vocabulary recognition performance. Specific improvements planned are :

- Addition of a tree-based lexicon.
- Stack decoding algorithm implementation.
- N-best output, to allow multi-pass decoding techniques to be employed.
- Implementation of a forced alignment feature, that can be used during training of speech recognition systems.
- ...

It is envisaged that the proposed improvements will complement the existing features of TODE, eventually providing users with a decoding toolkit containing efficient algorithms for different sized recognition tasks.

9 Conclusions

This report has described a flexible new decoder for continuous speech recognition called TODE (TOrch DEcoder). The basic problem of speech recognition was outlined, followed by a brief description of the fundamental dynamic programming and beam search algorithms that are utilised in TODE. The major software components of TODE were then discussed in detail and experimental results were presented comparing the performance of TODE to that of two popular decoders, HTK's HVite and Noway.

10 Acknowledgements

The author wishes to thank the Swiss National Science Foundation for supporting this work through the National Centre of Competence in Research (NCCR) on "Interactive Multimodal Information Management (IM2)".

References

[1] R. Collobert, S. Bengio, and J. Mariethoz. Torch: a modular machine learning software library. Technical Report IDIAP-RR-02-46, IDIAP, 2002.

[2] H. Ney and X. Aubert. Dynamic programming search strategies: from digit strings to large vocabulary word graphs. In C. Lee, F. Soong, and K. Paliwal, editors, *Automatic Speech and Speaker Recognition*, chapter 16, pages 385–411. Kluwer Academic Publishers, 1996.

[3] R. Bellman. *Dynamic Programming*. Princeton University Press, NJ, 1957.

[4] H. Bourlard, Y. Kamp, H. Ney, and C. Wellekens. Speaker dependent connected speech recognition via dynamic programming and statistical methods. In M. Schroeder, editor, *Speech and Speaker Recognition*, pages 115–148. Karger, Basel, 1985.

[5] H. Ney and S. Ortmanns. Dynamic programming search for continuous speech recognition. *IEEE Signal Processing Magazine*, 16(5):64–83, 1999.

[6] S. Young, G. Evermann, D. Kershaw, G. Moore, J. Odell, D. Ollason, V. Valtchev, and P. Woodland. *The HTK Book (for HTK Version 3.1).* Cambridge University, 2001.

[7] D. Paul. An efficient A* stack decoder algorithm for continuous speech recognition with a stochastic language model. In *Proceedings of ICASSP '92*, volume 1, pages 25–28, 1992.

[8] S. Renals and M. Hochberg. Decoder technology for connectionist large vocabulary speech recognition. Technical Report CUED/F-INFENG/TR.186, Cambridge University, 1995.

[9] S. Renals and M. Hochberg. Start-synchronous search for large vocabulary continuous speech recognition. *IEEE Transactions on Speech and Audio Processing*, 7:542–553, 1999.

[10] H. Ney, R. Haeb-Umbach, B. Tran, and M. Oerder. Improvements in beam-search for 10000-word continuous speech recognition. In *Proceedings of ICASSP '92*, volume 1, pages 9–12, 1992.