DeepSpeech: A Scalable Decoding System that Integrates Knowledge for Speech Recognition

Zifei Shan, Tianxin Zhao, Haowen Cao

Department of Computer Science, Stanford University
{zifei, tianxin, caohw}@stanford.edu

Abstract

We develop a scalable decoding system *DeepSpeech*, which flexibly integrates different levels of knowledge to decode a word lattice in speech recognition within a word-level CRF model.

DeepSpeech facilitates feature extraction, factor graph generation, and statistical learning and inference. It takes word lattice as input, perform feature extraction specified by developers, generate factor graphs based on descriptive rules, and perform learning and inference automatically. *DeepSpeech* is based on the scalable statistical inference engine DeepDive (http://deepdive.stanford.edu).

We integrate N-gram based linguistic features as well as some domain specific features. We train and evaluate our system on a dataset of broadcast news lattices, and obtain WER of 10.2%, which beats the baseline (Attila system) by a large margin. We also study a larger set of linguistic features with *DeepSpeech* and report their impact.

Index Terms: language model, advanced decoding

1. Introduction

1.1. Motivation

Speech recognition has been suffering from bad independence assumptions across different phases. Acoustic models are trained without syntactic and semantic knowledge beyond words, thus underestimated probabilities of correct words are hard to be captured by subsequent naive language models. We believe that eventually, joint inference on acoustic and language models is an especially promising future. As a first step towards this goal, we concentrate on advanced decoding in the language model phase. We propose to integrate more knowledge in decoding the word lattice, with the help of joint inference.

For state-of-the-art decoding approaches, it is hard to integrate sophisticated knowledge because of scalability reasons. However, with the cutting-edge researches on graph learning and sampling, we are now able to perform massive learning with a speed of millions of variables per second [9]. Weaponed with this, we use general factor graphs to do learning and inference on word lattices, which is able to integrate different kinds of knowledge including syntactic, semantic, context and higher-level knowledge.

Specifically, this paper tries to answer following questions:

1. How to jointly integrate different levels of knowledge in speech recognition?
2. Is it possible to have a real-time decoding system that approach oracle error rate of word lattices?
3. How to engineer the high-level features for language models in real-time ASR applications within current technology?

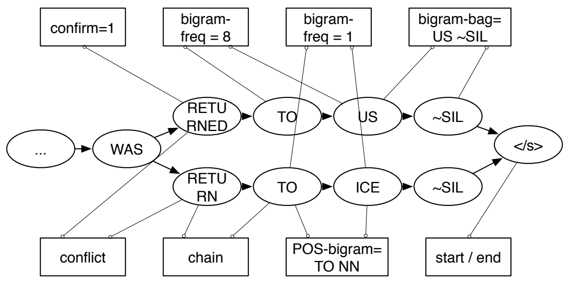


Figure 1: A Sample Factor Graph in DeepSpeech CRF Model

1.2. Previous Works

Previous works on integrating knowledge into speech recognition include different models: HMMs [2], Segmental CRFs [10], and Deep Neural Networks [3, 4]. Most works separate acoustic and language models, and it is hard for those works that focus on acoustic models to integrate higher-level language than lexicons. Most previous works on advanced decoding try to combine acoustic and language models by rescoring, without a rigorous probabilistic framework.

SCARF [10] is a platform that utilizes segmental conditional random fields. It provides a framework for developers to add their own linguistic features. However, the scalability of SCARF is questionable while it does not claim to be able to do real-time speech recognition.

1.3. Our approach

We propose a decoding system based on word-level Conditional Random Fields (CRFs) that is able to integrate various linguistic and domain-specific features.

CRFs are a type of general factor graphs. In our factor graph model, each **variable** is a word candidate in the lattice, and each **factor** is a feature or a domain-specific rule.

An example factor graph is shown in Figure 1. In this example, "was returned to us" and "was return to ice" are two different paths in the lattice. Each word in the lattice corresponds to a variable (shown as circles), and different factors might connect to them (shown as rectangles).

We obtain labels for variable-level training data by *distant supervision*: we match each lattice with its corresponding transcript, find all best paths in the lattice, and label all words on the best paths that matches a word in the transcript as *true*, other words as *false*.

As shown in Figure 1, the features we present in our system are such as: (1) word N-gram frequency; (2) bag of word N-grams (each N-gram itself is a feature); (3) a second "confirmatory" decoding made with an independent speech recognizer (which comes with the lattice data); (4) words around silence, etc.

DeepSpeech also enables developers to plug in more complicated features such as dependency paths, co-references, speaker-specific and contextual features.

1.4. Results

We got WER 10.2% on 152,251 broadcast news lattices with a simple feature set. (baseline 22.9%, oracle 2.1%) We finished training and testing in 70 minutes while the whole corpus is 400 hours, which indicates that our approach can be developed into a system that performs real-time decoding.

1.5. System

We propose to release *DeepSpeech* system [1] as an open-source platform for advanced decoding with flexible knowledge integration. Developers will be able to plug in their own extractors and apply the system to their own datasets. The system is built on a scalable inference engine *DeepDive* [6].

Using DeepSpeech provides following benefits: (1) easy extraction and integration of linguistic features; (2) simpler feature engineering loops; (3) a rigorous probabilistic framework.

2. Model

2.1. Problem Definition

Our problem is decoding a word lattice into one-best path, which is defined as follows: given an word lattice as input, which is a set of possible word sequences, the system outputs a word sequence with highest confidence.

2.2. Word-level Factor Graphs

We model the decoding problem as a word-level factor graph (a CRF).

Input to the system is a word lattice, which is a standard output by an acoustic speech recognition system.

Output of the system is a sequence of words (one-best path) for each lattice.

Variables are candidate words in the lattice. Among all variables, *query variables* are words in the test set, where we are not sure whether a candidate is correct or not; *evidence variables* are words in training set, where we can obtain true / false labels for each candidate word. After learning and inference, the system computes *marginal probabilities* for all query variables, which we will use to find the best path.

Factors are features or domain-specific rules that are related to candidate words. For example, the word bigram "of us" might have a feature indicating that it is a frequent bigram, where "of ice" might have a feature saying it is infrequent. Factors with different weights will be connected to the corresponding words. There might be more complex factors like constraints among conflicting candidates, chained candidates, coreferences, etc.

2.3. Distant Supervision to Obtain Training Data

On the factor graph we have factors with different weights to be learned. To train the model we need labeled data for evidence variables. However, although we have the transcript for each lattice in training set, there is no word-level ground truth indicating

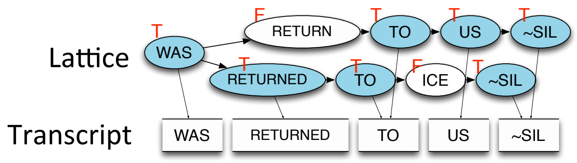


Figure 2: Distant Supervision

whether each candidate word is correct or not. To get training labels on a candidate level, we develop a *distant supervision* [5] metric to obtain labeled data in this setting.

Given a lattice and its transcript, we: (1) find all optimal paths in the lattice that matches the transcript with Dynamic Programming (DP), and (2) then label all matched words in all optimal paths as true, others as false.

In the distant supervision metric, we maximize the **number of matched words** on a path inside the lattice with the transcript. An alternative would be minimizing edit distance. The difference between these different objective functions are not studied here.

The distant supervision method is demonstrated in Figure 2. Specifically, in this lattice, both "WAS RETURN TO US ˜SIL" and "WAS RETURNED TO ICE ˜SIL" is a best path matching the transcript "WAS RETURNED TO US ˜SIL", since both of them have 4 matches.

As for the DP algorithm, denoting the number of candidates in lattice as N and number of words in transcript as M, then each candidate i memorizes a vector with length M, denoted as f. $f[i,j]$ is candidate i's maximum score matching up to transcript's position j. We perform DP according to the function below, in a topological ordering:

$$f[i,j] = \max_{i' \in Pred(i)} \begin{cases} f[i',j-1] + \mathbb{1}\{lattice[i] = transcript[j]\}, \\ f[i',j], \\ f[i,j-1] \end{cases}$$

2.4. Statistical Learning and Inference

After getting the factor graph with evidence, we perform statistical learning and inference. There are two steps in this procedure: (1) **learning:** performs *gradient descent* to calculate the values of weights for different factors. (2) **inference:** performs *gibbs sampling* to calculate the marginal probabilities of query variables. [8]

2.5. Finding Best Path on Factor Graphs

After learning and inference, each candidate word on the lattice gets a probability of being correct. Therefore the "best path" is well defined there: we want to find a path that minimizes the edit distance with the actual word sequence.

Here we formalize this problem: there exists an actual word sequence $C^* = \{a_1, a_2, ..., a_n\}$. Given an arbitrary path in the lattice $C = \{c_1, c_2, ..., c_n\}$, we have probabilities for each candidate to appear in the actual word sequence: $p_1, p_2, ..., p_n$. We want to minimize the edit distance of C and C^*, denoted as $Dist(C, C^*)$.

We want to minimize *edit distance*, which is the total number of insertions, deletions and substitutions. Consider a case in Figure 3, where there exists two paths $P1$ (with nodes SABCT) and $P2$ (with nodes SDT) in the lattice. Selecting $P1$ would suffer from an *expected number of insertions or substitutions* of 0.6, where each of A, B and C contributes $1 - 0.8 = 0.2$.

[1]Currently available at: `https://github.com/zifeishan/cs224s-deepSpeech/`

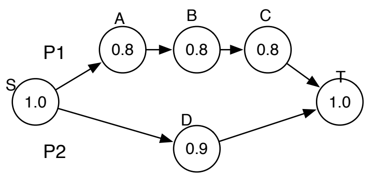


Figure 3: Finding Best Path with Inference Results

Similarly, selecting $P2$ only gives 0.1 expectation of insertion or substitution.

Let's look at *expected deletions*. Selecting $P1$ means not selecting node D, which gives expected number of deletions to be 0.9 (since D has the probability 0.9 to appear in actual sequence). Similarly, selecting $P2$ means not selecting A, B or C, which gives $0.8*3 = 2.4$ expected deletions.

In this way we can develop a dynamic programming metric to optimize edit distance. Equivalent to the above problem, we can add a punishment of -0.5 to each candidate: $p'_i = p_i - 0.5$, and select a path that optimizes $\sum_i p'_i$.

3. Architecture of DeepSpeech

We develop an end-to-end working system, DeepSpeech, which implements our data model. The system takes word lattice as input, performs user-defined feature extraction, factor graph generation, statistical learning and inference, and outputs the best path.

DeepSpeech is based on *DeepDive* [6], a scalable inference engine that facilitates feature extraction and generates factor graphs by a descriptive language. *DeepDive* has a high-throughput Gibbs sampler DimmWitted [9], which learns and samples at a speed of about 10 million variables per second on a laptop for our task.

The system architecture is demonstrated in Figure 4. We walk through each step of this data flow in this section.

3.1. Data Flow

3.1.1. Data preprocessing

In this step, DeepSpeech takes input data (lattices in raw format), runs preprocessing scripts on the data and loads them into a database. It also loads other data needed by the system, including transcripts for training, Google Ngram statistics, etc.

3.1.2. Feature Extraction

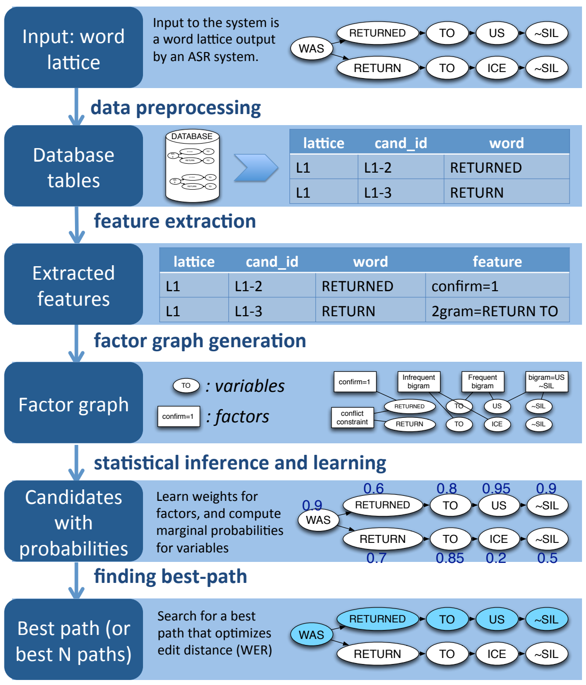
In this step, DeepSpeech extracts linguistic features by running "extractors" written by developers. Extractors are functionalities provided by DeepDive.

3.1.3. Factor Graph Generation

In the next step, DeepDive generates a factor graph. To tell the system how to generate it, developers use a SQL-like declarative language to specify *inference rules*, similar to Markov logic [7]. In inference rules, one can write first-order logic rules with **weights**, which intuitively model our confidence in a rule.

3.1.4. Statistical Inference and Learning

This step is automatically performed by DeepDive on the generated factor graph. In learning, the values of weights specified in



lattice	cand_id	word
L1	L1-2	RETURNED
L1	L1-3	RETURN

lattice	cand_id	word	feature
L1	L1-2	RETURNED	confirm=1
L1	L1-3	RETURN	2gram=RETURN TO

Figure 4: Architecture of DeepSpeech

inference rules are calculated. In inference, marginal probabilities of variables are computed.

3.1.5. Finding Best Path

After inference, DeepSpeech performs a search for a best path that optimizes the edit distance, with the algorithm discussed in Section 2.5. The system then outputs the best path it finds.

3.2. DeepSpeech Features

Our DeepSpeech system provides following benefits, compared to other systems like SCARF [10].

3.2.1. Easy Extraction and Integration of linguistic features

DeepSpeech provides a framework for developers to extract and integrate high-level features by writing MapReduce-like functions in SQL and python. This functionality is provided by DeepDive's feature extraction and factor graph generation frameworks.

3.2.2. Simpler Feature Engineering

Powered by DeepDive, DeepSpeech enables developers to conduct systematic feature engineering iteratively. Same as the "E3-loop" demonstrated in [1], developers go through Explore-Extract-Evaluate loops to continuously improve the system: (1) explore results and do error analysis, (2) improve extractors to get better features, and (3) rerun the system to get new results.

3.2.3. Rigorous Probabilistic Framework

Unlike traditional speech recognition systems that uses ad-hoc methods to combine the scores given by a language model and an acoustic model, DeepSpeech provides a rigorous probabilistic framework: every probability it predicts has the strict probabilistic meaning, which is "the likelihood of the word candidate to be in the actual word sequence". The probabilities are well-calibrated, which means that it is supported by data. With this probabilistic framework, finding the best-path (or N-best paths) is straightforward.

4. Experiments

In this section, we report initial experiments we conduct using DeepSpeech. We train and evaluate the system on a large-scale dataset, and compare the results with a baseline system and oracle (lattice optimal) error rate. We further look into the impact of different features.

4.1. System Evaluation

4.1.1. Datasets

We train and test on broadcast news lattices (LDC2011T06, 152,251 lattices). We holdout 50% of training set for testing.

The oracle error rate (lattice optimal WER) is 2.1% on this dataset. Our baseline is the one-best word detections from the Attila system, provided by LDC2011T06 dataset.

4.1.2. Features

After initial feature engineering, we use following features:

1. The "confirmatory" decoding made with an independent speech recognizer.
2. Unigram and bigram frequency in Google Ngram: For each candidate word N-gram, we take its frequency (log-scale) in Google Ngram dataset as a feature that indicates whether it is a valid word / phrase. We skip disfluencies such as "um", "uh" and silence in this feature.
3. Bag of word bigrams. Each bag of bigrams itself is a different feature (e.g. "we are", "but uh"). This is a very sparse feature, but with the large dataset and proper regularization, it performs well.
4. All words around silence. We take the bigram of a word and a silence in speech, to capture what words are likely to come before or after a silence.

4.1.3. Results

We use the standard tool SCLITE for scoring. We evaluate our system, a baseline system (Attila), and lattice oracle (optimal) word error rate. The word error rate includes substitutions, deletions and insertions.

The results are shown in Table 4.1.3.

System	Corr	Sub	Del	Ins	Err	S.Err
Baseline	77.8	5.4	16.8	0.6	22.9	96.9
DeepSpeech	92.0	3.6	4.3	2.2	10.2	75.7
Oracle	99.9	0.0	0.1	2.0	2.1	50.8

Table 1: Experiment Results

About performance: DeepSpeech runs 70 minutes for training and testing, while this dataset is about 400 hours of speech. This indicates that DeepSpeech can potentially make a real-time decoding system that integrates high level knowledge "for free".

4.2. Feature Exploration

In this section, we enumerate a larger set of different features we implement in DeepDive, and test on each feature's impact.

For dataset, we use a subsample of the broadcast lattice dataset. We only takes 1,000 lattices (speeches) and split them half-half for training and testing.

4.2.1. List of features

Features we implement are listed below. Some are discussed in the above section while some are newly introduced. Some of these features are also shown in Figure 1.

1. (*Ngram-freq*) Google Ngram frequency. (N=1,2,...)
2. (*confirm*) The "confirmatory" decoding made with an independent speech recognizer, provided by LDC dataset.
3. (*start-end*) Start and end marks of sentence (`<s>` and `</s>`): each sentence is starting and ending with a special mark. We add a factor to a candidate word if it is this special mark.
4. (*Ngram-bag*) Bag of word N-grams. Each bag of N-grams itself is a different feature. (N=1,2)
5. (*Ngram-stopword*) Any N-gram containing a stop-word (stopwords include silence). (N=2,3)
6. (*Ngram-silence*) All words around silence, which is the N-gram containing a `~SIL` mark. (we can see that $7 \in 6 \in 5$) (N=2)
7. (*conflict*) Conflict constraint. This is a CRF rule that adds a factor to connect two variables: candidates that overlap in time cannot be both true.
8. (*chain*) Chaining candidates on same paths. This is a linear-chain CRF rule: candidates on a same path should be true at same time.
9. (*pos-Ngram*) POS N-gram and trigram. This rule takes all N-grams and trigrams of candidate as a feature. POS tags are tagged for each word independently with a one-best tagger, without sentence structure. (N=2,3,5)

4.2.2. Experiments Protocol

We increment features and observe impact for each feature.

We start from unigram frequency, then add bigram frequency and trigram frequency. Then we try using *only* the "confirm" feature, adding start-end feature onto it, adding unigrams and bigrams, then adding bigram-silence, bigram-stopword, and bigram-bag (features get more and more sparse). We further add conflict rule and "chain" rule. At last we add larger language models like bags of word 1grams, POS 5grams, and stop word trigrams.

4.2.3. Experiment results

We report the observed word error rate (WER) and sentence error rate (SER) in Figure 5.

Specifically, we take the following lessons:

(1) Unigram frequency itself does not work, since most ASR softwares tries to give valid words in dictionary.

(2) 1/2/3gram frequency feature (WER 22.3%) is not as good as "confirm" feature alone (WER 13.6%). This indicates that a weak language model is not as competitive as a good second-confirmatory system (ensemble wins).

(3) Sparse bag-of-Ngram feature helps. By adding feature *bigram-silence*, *bigram-stopword* and *bigram-bag* incrementally, we observe continuous decrease in WER (11.3%, 10.8%, 10.1%). This indicates that our system is able to make reasonable regularization and handle sparsity well.

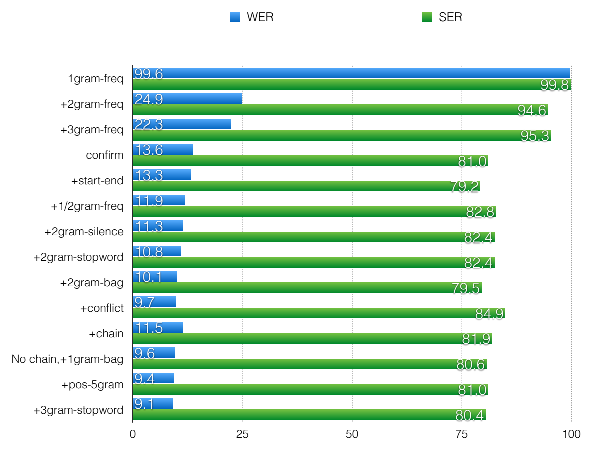


Figure 5: Feature Exploration Results

(4) The "conflict" constraint decreases WER, but increases SER; while the "chain" constraint increases WER, but decreases SER.

(5) The larger language model we feed DeepSpeech, the better results it gets. When we feed stopword trigram and POS 5gram, it gets WER of only 9.1%.

5. Future Work

In the future, we will continue integrating more knowledge into DeepSpeech, including high level features like coreferences, dependency paths and knowledge base, and also integrate low-level acoustic features.

We will further look into candidate generation with high-level knowledge. For now we still treat the decoding problem as a classification problem, which is picking from existing candidates. However with linguistic, contextual and knowledge-base knowledge, we might be able to generate new candidates into the lattice, and this might enable us to beat oracle error rate.

It would be also interesting to ensemble different speech recognition systems to obtain larger lattices, and try to learn a model that aggregates different systems and enables inter-system corrections.

6. Conclusion

We present a model and a system that is able to integrate different levels of knowledge to decode word lattices generated by ASR systems.

Our model is word-level conditional random fields, where word candidates are variables and features and domain-specific rules are factors. To obtain word-level training data in the factor graph, we distantly supervise the system by matching lattice words to transcripts, and label matched words on all best paths as true (others as false). Statistical learning and inference on these factor graphs give us rigorous probabilities that we can use to find best paths as outputs.

We present a system *DeepSpeech* that implements the above model. Our system is built on a scalable inference engine *DeepDive*, and is able to train and test on 400-hour speech data within 70 minutes.
We propose to improve and release the system, with which developers will be able to add their own linguistic features easily, and conduct advanced decoding on their own datasets in real time.

We conduct experiments to evaluate DeepSpeech and explore different features. Evaluating with 150K broadcast news lattices, we get WER of 10.2% where the baseline Attila system gets WER up to 22.9%.

In our study of different features, we have several insights including (1) word unigram and POS features does not help much; (2) a second confirmatory decoding system works better than a word-Ngram based language model; (3) DeepSpeech is able to utilize very sparse bag-of-Ngram features; (4) some CRF rules has different impact on WER and SER.

In the future we will release DeepSpeech system, integrate more high-level knowledge to approach oracle WER, study candidate generation methods to go over oracle WER, and try an ensemble system of different speech engines.

7. Acknowledgements

The authors would like to acknowledge Professor Christopher Re, for his support to this project and the DeepDive platform under rapid development.

The authors would also like to acknowledge Professor Dan Jurafsky for his encouragements and initial discussions regarding the project ideas.

Thanks to Andrew Mass for iterating the project ideas, giving hands-on suggestions and feedback. Thanks to Ce Zhang for his inspiring ideas for candidate generation. Thanks to Natalia Silveira and Stanford NLP group for providing us the datasets. Thanks to Juergen Fesslmeier and Ana Maria Carrano from *Voyz.es*, for showing interest in the poster session and follow-up discussions.

8. References

[1] M. Anderson, D. Antenucci, V. Bittorf, M. Burgess, M. J. Cafarella, A. Kumar, F. Niu, Y. Park, C. Ré, and C. Zhang. Brainwash: A data system for feature engineering. In *CIDR*, 2013.

[2] L. R. Bahl, F. Jelinek, and R. Mercer. A maximum likelihood approach to continuous speech recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (2):179–190, 1983.

[3] G. E. Dahl, D. Yu, L. Deng, and A. Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(1):30–42, 2012.

[4] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6):82–97, 2012.

[5] M. Mintz, S. Bills, R. Snow, and D. Jurafsky. Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pages 1003–1011. Association for Computational Linguistics, 2009.

[6] C. Ré. Deepdive, by hazy research group. `http://deepdive.stanford.edu/`.

[7] M. Richardson and P. Domingos. Markov logic networks. *Machine learning*, 62(1-2):107–136, 2006.

[8] C. Zhang and C. Ré. Towards high-throughput gibbs sampling at scale: A study across storage managers. In *Proceedings of the 2013 international conference on Management of data*, pages 397–408. ACM, 2013.

[9] C. Zhang and C. Ré. Dimmwitted: A study of main-memory statistical analytics. *CoRR*, abs/1403.7550, 2014.

[10] G. Zweig and P. Nguyen. Scarf: a segmental conditional random field toolkit for speech recognition. In *INTERSPEECH*, pages 2858–2861, 2010.