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

DeepSpeech flexibly integrates different levels of knowledge to decode a word lattice in speech recognition within a word-level CRF model, in an interpretable manner.

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).

Index Terms: language model, advanced decoding

1. Introduction

Problem: How to jointly integrate different levels of knowledge in speech recognition?

Solution: Decoding based on Conditional Random Fields that integrates various features.

Results: Got WER 10.2% on 150k broadcast news lattices in near real-time, with a simple feature set. (baseline 22.9%, oracle 2.1%)

Future: Candidate generation with linguistic knowledge might beat oracle error rate; joint inference on acoustic and language models

2. The Architecture of DeepSpeech

Architecture: See Figure 1.

2.1. Distant supervision to obtain training data

We use distant supervision techniques to get training labels on candidate level: given a lattice and its transcript, we:

1. Find optimal paths in the lattice that matches the transcript with Dynamic Programming
2. Label all matched words in all optimal paths as true, others as false

See Figure 2

3. Algorithms

4. Evaluation

4.1. Experimental Setup

4.1.1. Features

After initial feature engineering, our current system implements following features:

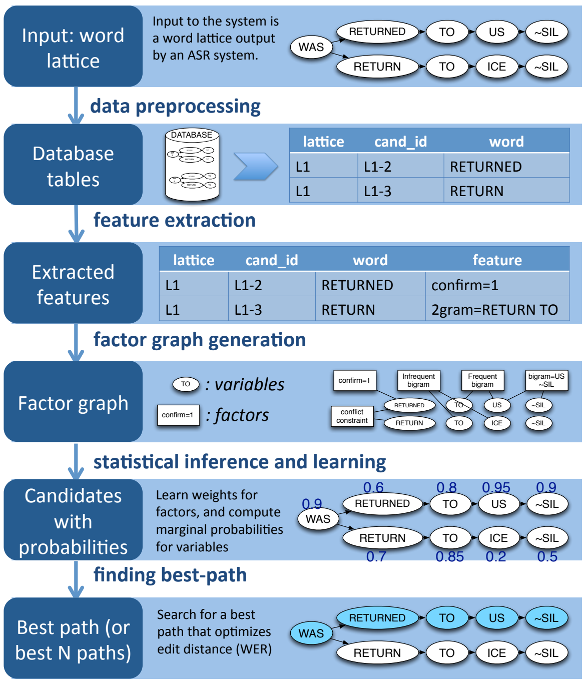
Input: word lattice
Input to the system is a word lattice output by an ASR system.
WAS
RETURNED
TO
US
~SIL
RETURN
TO
ICE
~SIL
data preprocessing
Database tables
DATABASE
lattice | cand_id | word
L1 | L1-2 | RETURNED
L1 | L1-3 | RETURN
feature extraction
Extracted features
lattice | cand_id | word | feature
L1 | L1-2 | RETURNED | confirm=1
L1 | L1-3 | RETURN | 2gram=RETURN TO
factor graph generation
Factor graph
TO : variables
confirm=1 : factors
confirm=1
Infrequent bigram
Frequent bigram
bigram=US ~SIL
conflict constraint
statistical inference and learning
Candidates with probabilities
Learn weights for factors, and compute marginal probabilities for variables
0.9
0.6
0.8
0.95
0.9
0.7
0.85
0.2
0.5
finding best-path
Best path (or best N paths)
Search for a best path that optimizes edit distance (WER)

Figure 1: Architecture of DeepSpeech

1. Unigram and bigram frequency in Google Ngram. (skip "silence")
2. All bigrams around "silence"
3. POS tag 2gram and 3gram
4. Candidates that overlap in time cannot be both true (a CRF rule that indicates constraint)
5. Candidates on a same path should be true at same time (a linear-chain CRF rule)

TODO: Next steps: co-reference features and candidate generation

4.1.2. Datasets

We train and test on broadcast news lattices (LDC2011T06, 150k lattices). We holdout 50% of training set for testing.

4.2. Results

We use SCLITE for scoring. We evaluate a baseline system (Attlia), DeepSpeech and lattice oracle (optimal) error rate.

Performance: DeepSpeech runs 70min for training and testing, while this dataset is ˜400 hours of speech (real-time!)

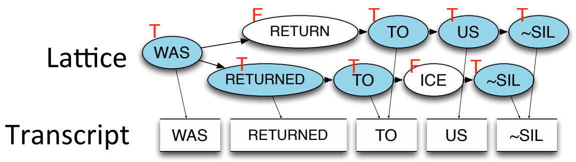

Figure 2: Distant Supervision

System	Corr	Sub	Del	Ins	Err	S.Err
Baseline	77.8	5.4	16.8	0.6	22.9	96.9
DeepSpeech	92	3.6	4.3	2.2	10.2	75.7
Oracle	99.9	0	0.1	2	2.1	50.8

Table 1: Experiment Results

4.3. Feature Engineering

TODO show how good is each feature

5. DeepSpeech Features

5.1. Extract & Integrate linguistic features

DeepSpeech provides a framework for developers to extract and integrate high-level features

Here is how developers can easily plug-in a corefence feature “extractor” to DeepSpeech:

SQL:

Define a SQL query to generate all pairs of candidate words that appear in the same lattice, and pair it with a python function.

```
SELECT t0.CID, t0.TEXT,
       t1.CID, t1.TEXT
FROM   candidate t0,
       candidate t1
WHERE  t0.LID = t1.LID
USEPYTHON pyfunc
```

PYTHON:

We write a Python function to process all phrase pairs and identify coreferent pairs.

```
def pyfunc(c1, t1, c2, t2):
    if edit_dist(t1, t2) < 2:
        emit(Coref, c1, c2)
```

5.2. Simpler Feature Engineering

TODO brainwash paper

5.3. Rigorous Probabilistic Framework

TODO paper

6. Related Work

[1].

7. Future Work

8. Conclusion

9. Acknowledgements

The authors would like to acknowledge. . .

10. References

[1] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.