

음성 인식 실습 - Kaldi

(주) 보이스프린트

대표 김광호

강사 소개

● 김광호

- 현재 : (주) 보이스프린트 대표이사 (음성인식 분야)
- 2016.12 ~ 2018.03 : (주) 에이젠글로벌 수석연구원 (금융AI 분야)
- 2016.03 ~ 2016.11 : (주) 티앤블루랩 책임연구원 (음악검색 분야)
- 2010.03 ~ 2016.02 : 서강대학교 박사과정 (음성인식 전공)
- 2008.03 ~ 2010.02 : 서강대학교 석사과정 (음성인식 전공)

Kaldi: Speech Recognition Software

● C++ 기반의 오픈소스 음성인식 툴

(License : Apache License v2.0)

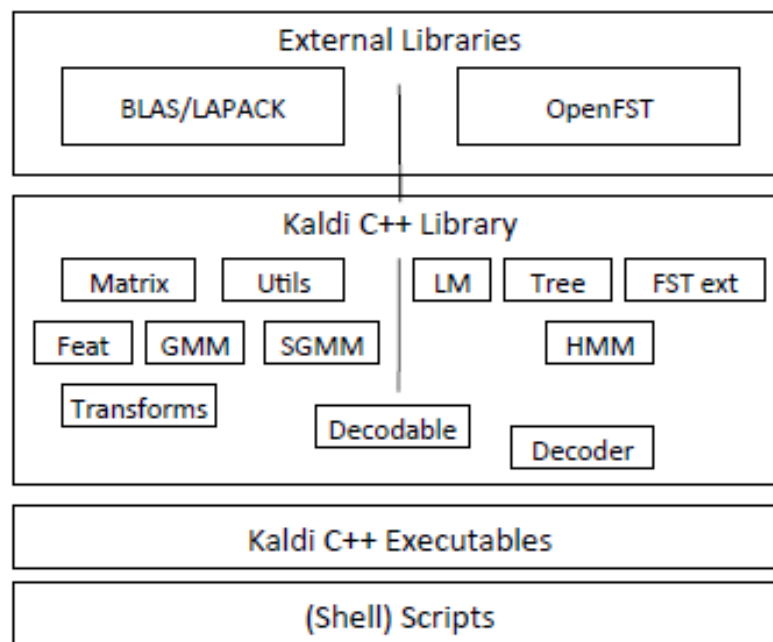
- C++ 기반의 객체 지향 형태로 구현되어 있어, 새로운 입력 및 내부 구조 변경에 대해 확장 가능성 있도록 구현함
- 디코딩 과정에서 WFST 활용
 - Code-level integration with Finite State Transducers (FSTs)
- LDC에서 제공하는 음성 코퍼스를 이용한 Recipe를 제공
 - 대표적 영어 연속 음성인식 Recipe
 - Resource Management (RM) 코퍼스 : 미국 국방성에서 지원한 소량의 학습 데이터를 이용한 영어 연속 음성인식 도메인 음성 코퍼스
- 새로운 SW 개발시 소스코드 공개 의무 없으며, 상업적 이용에 제한이 없음

Kaldi: Speech Recognition Software

- C++ 기반의 오픈소스 음성인식 툴

(License : Apache License v2.0)

- Kaldi는 외부 라이브러리와 C++ 클래스 라이브러리를 스크립트에서 호출하여 사용하는 방식



Kaldi: Speech Recognition Software

- ◉ Kaldi 공식 사이트
 - <http://kaldi-asr.org/>
- ◉ Kaldi git 주소
 - <https://github.com/kaldi-asr/kaldi>
- ◉ Kaldi QnA Forum
 - <https://groups.google.com/forum/#!forum/kaldi-help>
- ◉ Daniel Povey's homepage
 - <http://danielpovey.com/>

Kaldi: Speech Recognition Software

- Kaldi 이전의 기존 음성인식 툴 문제점
- HTK: Hidden Markov Model (HMM)을 구현
 - HMM이 적용되는 알고리즘에서 사용됨
 - 인식 성능은 높았으나, 응답 속도가 100RT 수준 (90년대 초)
- Sphinx4 : CMU대학에서 연구 목적으로 Java 언어로 구현
 - 실 환경에서 사용할 수 있도록 하는 지원이 부족

Librispeech Corpus

- Librispeech corpus는 LibriVox 를 source로 하여 음성인식에 맞게 구성한 dataset
 - LibriVox는 public domain audiobooks 으로서
 - 누구나 지원하여 책읽기를 수행한 후 업로드 할 수 있으며 무료로 다운로드 가능

Free public domain audiobooks

— Read by volunteers from around the world. —

Read

LibriVox audiobooks are read by volunteers from all over the world. Perhaps you would like to join us?

VOLUNTEER

Listen

LibriVox audiobooks are free for anyone to listen to, on their computers, iPods or other mobile device, or to burn onto a CD.

CATALOG

Librispeech Corpus

- ◉ Librispeech corpus의 data 구성
 - dev-clean, test-clean
 - Clean speech로 구성된 development and test set
 - Dev-other, test-other
 - Challenging to recognize speech로 구성된 development and test set
 - Train-clean-100
 - Clean speech 100시간
 - Train-clean-360
 - Clean speech 360시간
 - Train-other-500
 - Challenging to recognize speech 500시간

Kaldi 디렉토리 구조 설명

● Kaldi 디렉토리 구조

(현재 위치 : ~/kaldi-trunk/egs/librispeech/s5)

```
RESULTS  cmd.sh  conf  local  path.sh  run.sh  steps  utils
```

- 학습을 수행하기 이전에는 conf, local, steps, utils 디렉토리만 존재
 - conf
 - Kaldi에서 제공하는 명령어에 입력되는 옵션 parameter 값에 대한 파일 디렉토리
 - local
 - 각 corpus 별 예제 스크립트에 맞는 shell 및 perl script 파일 디렉토리
 - steps, utils
 - 모든 corpus에서 공통으로 사용되는 shell 및 perl script 파일 디렉토리로, WSJ 예제의 step와 utils 디렉토리에 대한 symbolic link

Kaldi 디렉토리 구조 설명

● Kaldi 디렉토리 구조

(현재 위치 : ~/kaldi-trunk/egs/librispeech/s5)

```
RESULTS  cmd.sh  conf  local  path.sh  run.sh  steps  utils
```

- 학습을 수행하기 이후에는 data, mfcc, exp 디렉토리가 생성
 - data
 - Kaldi 모델 학습에 사용되는 input data 파일 디렉토리
 - mfcc
 - 특징 추출한 결과가 저장되는 디렉토리
 - exp
 - 다양한 모델별로 음향 모델을 저장하고 인식 그래프 및 인식 결과를 저장하는 디렉토리

Kaldi 디렉토리 구조 설명

◉ cmd.sh (현재 위치 : ~/kaldi-trunk/egs/rm/s5)

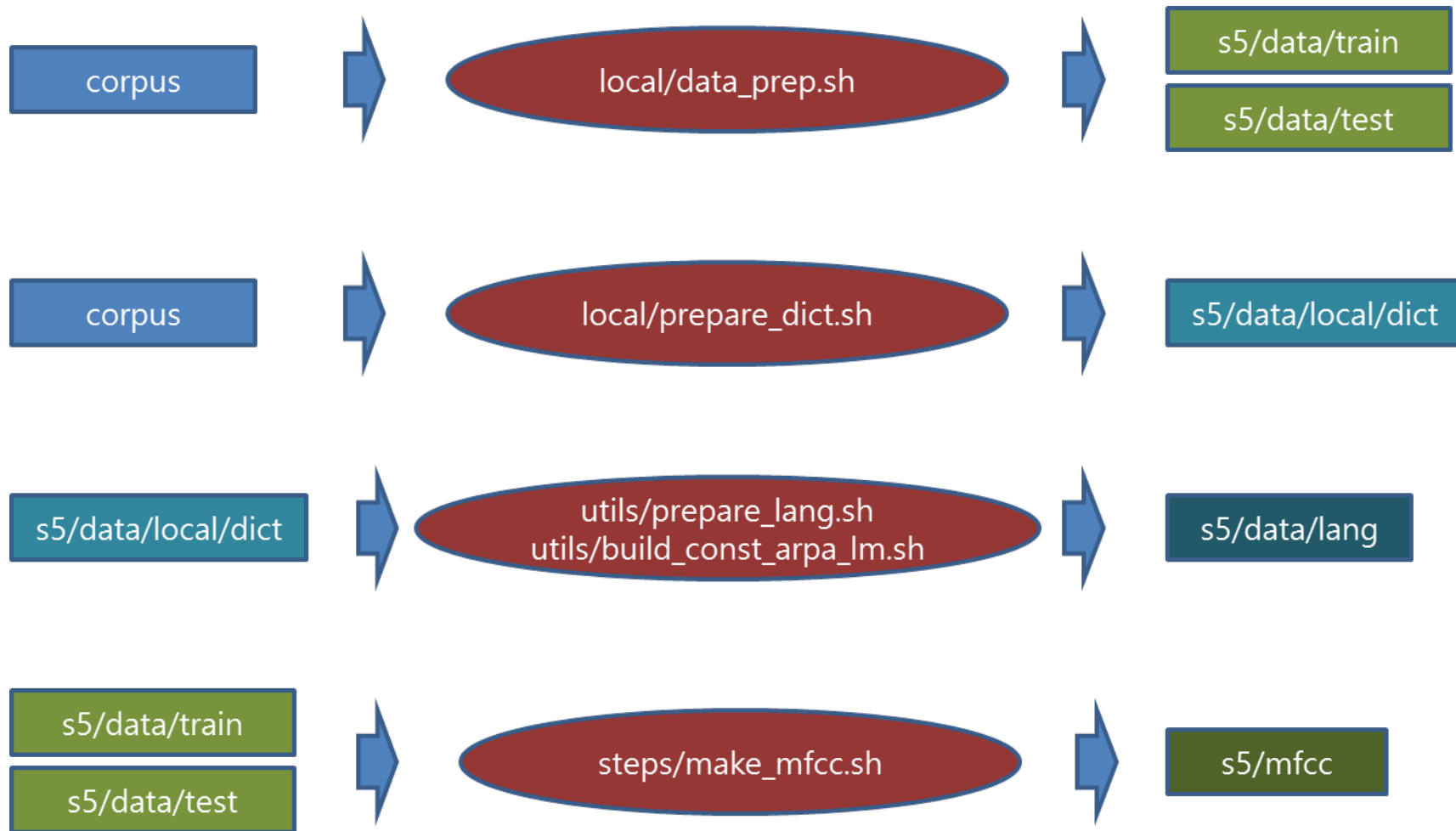
- Kaldi는 Oracle GridEngine이 설치된 여러 대의 서버 상에서 학습을 수행하는 것도 지원
- cmd.sh 파일에서는 이를 지원할지의 여부를 결정
- 실습에서는 단일 서버 상에서 학습을 진행
 - cmd.sh 의 모든 내용을 주석 처리하고, 아래의 내용을 작성

```
train_cmd="run.pl"  
decode_cmd="run.pl"  
cuda_cmd="run.pl"
```

- 실제 DNN-HMM 기반 음향모델 학습 시 GridEngine이 설치된 여러 대의 서버 상에서 학습하는 것을 권장

```
train_cmd="queue.pl -l arch=*64"  
decode_cmd="queue.pl -l arch=*64"  
cuda_cmd="queue.pl -l arch=*64 -l gpu=1"
```

특징 추출까지의 과정 (run.sh)



특징 추출까지의 과정 (run.sh)

● local/data_prep.sh

- Usage
 - ./local/data_prep.sh <srd-dir> <dst-dir>

```
for part in dev-clean test-clean dev-other test-other train-clean-100; do
# use underscore-separated names in data directories.
local/data_prep.sh $data/LibriSpeech/$part data/$(echo $part | sed s/-/_/g)
```

- data 디렉토리의 train-clean-100, dev-clean, test-clean 등의 디렉토리별로 dict 디렉토리를 구성하기 위한 shell script
- librispeech corpus의 하부 디렉토리별로 shell script를 수행함

특징 추출까지의 과정 (run.sh)

● local/data_prep.sh 결과물

- s5/data/train-clean-100
- s5/data/dev-clean
- s5/data/dev-other
- s5/data/test-clean
- s5/data/test-other
- 위의 디렉토리 별로 다음 파일들이 생성 됨
 - spk2utt
 - Speaker to utterance (화자가 발성한 utterance 정보)
 - utt2spk
 - Utterance to speaker (각 utterance를 발성한 화자 정보)

특징 추출까지의 과정 (run.sh)

● local/data_prep.sh 결과물

- s5/data/train-clean-100
- s5/data/dev-clean
- s5/data/dev-other
- s5/data/test-clean
- s5/data/test-other
- 위의 디렉토리 별로 다음 파일들이 생성 됨
 - text
 - First column - Utterance에 대한 key
 - Second column - 정답 script

```
103-1240-0000 CHAPTER ONE MISSUS RACHEL LYNDE IS SURPRISED MISSUS RACHEL LYNDE LIVED JUST WHERE  
ARDROPS AND TRAVERSED BY A BROOK  
103-1240-0001 THAT HAD ITS SOURCE AWAY BACK IN THE WOODS OF THE OLD CUTHBERT PLACE IT WAS REPUTE  
SECRETS OF POOL AND CASCADE BUT BY THE TIME IT REACHED LYNDE'S HOLLOW IT WAS A QUIET WELL CONDU  
103-1240-0002 FOR NOT EVEN A BROOK COULD RUN PAST MISSUS RACHEL LYNDE'S DOOR WITHOUT DUE REGARD  
WINDOW KEEPING A SHARP EYE ON EVERYTHING THAT PASSED FROM BROOKS AND CHILDREN UP
```

특징 추출까지의 과정 (run.sh)

● local/data_prep.sh 결과물

- s5/data/train-clean-100
- s5/data/dev-clean
- s5/data/dev-other
- s5/data/test-clean
- s5/data/test-other
- 위의 디렉토리 별로 다음 파일들이 생성 됨
 - wav.scp
 - First column - Utterance에 대한 key
 - Second column - 정답 script에 대한 발성 wav 파일 경로
 - Kaldi에서는 학습자료로 wav 파일을 사용
 - Little-endian
 - 16bit, mono, 16kHz

특징 추출까지의 과정 (run.sh)

● local/data_prep.sh 결과물

- s5/data/train-clean-100
- s5/data/dev-clean
- s5/data/dev-other
- s5/data/test-clean
- s5/data/test-other
- 위의 디렉토리 별로 다음 파일들이 생성 됨
 - wav.scp
 - librispeech는 wav 파일 포맷이 아닌 flac 포맷으로 저장되어 있음
 - wav 파일 경로가 아닌 flac을 이용한 decode결과를 pipe로 넘겨주도록 script

```
103-1240-0000 flac -c -d -s /home/splab/DATA/LibriSpeech/train-clean-100/103/1240/103-1240-0000.flac |
103-1240-0001 flac -c -d -s /home/splab/DATA/LibriSpeech/train-clean-100/103/1240/103-1240-0001.flac |
103-1240-0002 flac -c -d -s /home/splab/DATA/LibriSpeech/train-clean-100/103/1240/103-1240-0002.flac |
103-1240-0003 flac -c -d -s /home/splab/DATA/LibriSpeech/train-clean-100/103/1240/103-1240-0003.flac |
103-1240-0004 flac -c -d -s /home/splab/DATA/LibriSpeech/train-clean-100/103/1240/103-1240-0004.flac |
103-1240-0005 flac -c -d -s /home/splab/DATA/LibriSpeech/train-clean-100/103/1240/103-1240-0005.flac |
```

특징 추출까지의 과정 (run.sh)

● local/data_prep.sh 결과물

- s5/data/train-clean-100
- s5/data/dev-clean
- s5/data/dev-other
- s5/data/test-clean
- s5/data/test-other
- 위의 디렉토리 별로 다음 파일들이 생성 됨
 - spk2gender
 - 화자의 성별을 기입 (female or male)

```
103-1240 f
103-1241 f
1034-121119 m
1040-133433 m
1069-133699 f
1069-133709 f
1081-125237 m
1081-128618 m
```

특징 추출까지의 과정 (run.sh)

● local/data_prep.sh 결과물

- s5/data/train-clean-100
- s5/data/dev-clean
- s5/data/dev-other
- s5/data/test-clean
- s5/data/test-other
- 위의 디렉토리 별로 다음 파일들이 생성 됨
 - utt2dur
 - Utterance의 재생시간을 기입
 - nnet3 / chain 모델에 필요
 - nnet2 까지 수행하는데에는 필요치 않음

특징 추출까지의 과정 (run.sh)

● local/prepare_dict.sh

- Usage

- ./local/prepare_dict.sh <lm-dir> <g2p-model-dir> <dst-dir>

```
local/prepare_dict.sh --stage 0 --nj 30 --cmd "$train_cmd" \
  data/local/lm data/local/lm data/local/dict_nosp
```

- cmudict 및 sequitur g2p를 이용하여 phoneme set, lexicon 등을 구성하기 위한 shell script
 - 다음과 같은 파일이 생성됨
 - lexicon.txt
 - lexiconp.txt
 - nonsilence_phones.txt
 - silence_phones.txt
 - optional_silence.txt
 - extra_question.txt

특징 추출까지의 과정 (run.sh)

● local/prepare_dict.sh 의 결과물

- s5/data/local/dict_nosp 디렉토리 생성
 - lexicon.txt
 - First column – Word (sort 되어 있음)
 - Second column
 - Phoneme sequence
 - Grapheme-to-Phoneme(G2P) 변환을 통해 주어진 영어 단어에 대한 발음 set
 - !SIL – silence
 - <SPOKEN_NOISE> - 발성 외 구간
 - <UNK> - 모든 out of vocabulary를 맵핑

```
A AH0
A EY1
A'S EY1 AH0 Z
A'BODY EY1 B AA2 D IY0
A'COURT EY1 K AO2 R T
```

특징 추출까지의 과정 (run.sh)

● local/prepare_dict.sh 의 결과물

- s5/data/local/dict_nosp 디렉토리 생성
 - lexiconp.txt
 - Kaldi에서는 lexicon.txt를 기반으로 lexiconp.txt를 생성함
 - 만약 lexicon.txt와 lexiconp.txt가 동시에 존재 할 경우 lexiconp.txt를 이용하게 됨
 - lexicon.txt와의 차이점
 - Second column이 가중치 정보
 - 동일 영어 단어임에도 불구하고, 다른 phoneme sequence가 있을 경우, 가중치의 합이 1.0이 되도록 가중치 를 부여함

```
A 1.0 AH0
A 1.0 EY1
A''S 1.0 EY1 AH0 Z
A'BODY 1.0 EY1 B AA2 D IY0
A'COURT 1.0 EY1 K AO2 R T
```

특징 추출까지의 과정 (run.sh)

- local/prepare_dict.sh 의 결과물
 - s5/data/local/dict_nosp 디렉토리 생성
 - nonsilence_phones.txt
 - Non-silence phone에 대한 list
 - lexicon.txt에서 사용된 각 phoneme들에 대한 list
 - silence_phones.txt
 - silence phone에 대한 list
 - optional_silence.txt
 - lexicon.txt의 SIL에 해당되는 silence에 대한 list

특징 추출까지의 과정 (run.sh)

● utils/prepare_lang.sh

- Usage

```
usage: utils/prepare_lang.sh <dict-src-dir> <oov-dict-entry> <tmp-dir> <lang-dir>
e.g.: utils/prepare_lang.sh data/local/dict <SPOKEN_NOISE> data/local/lang data/lang
<dict-src-dir> should contain the following files:
  extra_questions.txt  lexicon.txt  nonsilence_phones.txt  optional_silence.txt  silence_phones.txt
See http://kaldi-asr.org/doc/data\_prep.html#data\_prep\_lang\_creating for more info.
```

- Lexicon에 대한 fst인 L.fst 및 L_disambiguation.fst를 생성하고, lexicon.txt에서 선언된 각 영어 단어에 대한 indexing 작업을 수행
- run.sh에서 선언된 prepare_lang.sh 사용법

```
utils/prepare_lang.sh data/local/dict_nosp \
  "<UNK>" data/local/lang_tmp_nosp data/lang_nosp
```

- Out-of-vocabulary를 <UNK>로 선언하여 사용하고 있음
- data/local/dict-nosp 디렉토리에 data_prep.sh 의 결과 파일이 존재하여야 함
- 최종 결과물은 s5/data/lang_nosp 에 생성됨

특징 추출까지의 과정 (run.sh)

● utils/build_const_arpa_lm.sh

- Usage

```
Usage:
  utils/build_const_arpa_lm.sh [options] <arpa-lm-path> <old-lang-dir> <new-lang-dir>
e.g.:
  utils/build_const_arpa_lm.sh data/local/lm/3-gram.full.arpa.gz data/lang/ data/lang_test_tgmed
```

- arpa format language model을 이용하여 G.fst가 생성된 새로운 lang directory를 생성
- run.sh에서 선언된 build_const_arpa_lm.sh 사용법

```
# Create ConstArpaLm format language model for full 3-gram and 4-gram LMs
utils/build_const_arpa_lm.sh data/local/lm/lm_tglarge.arpa.gz \
  data/lang_nosp data/lang_nosp_test_tglarge
utils/build_const_arpa_lm.sh data/local/lm/lm_fglarge.arpa.gz \
  data/lang_nosp data/lang_nosp_test_fglarge
```

- tri-gram 언어모델 및 four-gram 언어모델 생성
- 음향모델을 학습한 후에 다양한 언어모델을 적용한 디코딩 그래프를 생성할 수 있음

특징 추출까지의 과정 (run.sh)

- Kaldi에서는 MFCC, PLP, Filterbank 특징 추출을 제공하고 있음
 - steps 디렉토리에 다음과 같은 script 제공
 - MFCC
 - make_mfcc.sh, make_mfcc_pitch.sh, make_mfcc_pitch_online.sh
 - PLP
 - make_plp.sh, make_plp_pitch.sh
 - Filterbank
 - make_fbank.sh, make_fbank_pitch.sh
 - 본 실습에서는 MFCC 특징 추출 수행

특징 추출까지의 과정 (run.sh)

● steps/make_mfcc.sh

- wav 파일 혹은 wav pipe로부터 MFCC 특징 추출 과정을 수행

● steps/compute_cmvn_stats.sh

- 추출된 MFCC 특징으로부터 Cepstral Mean and Variance Normalization (CMVN) 정보를 추출하는 과정 수행

```
for part in dev_clean test_clean dev_other test_other train_clean_100; do
  steps/make_mfcc.sh --cmd "$train_cmd" --nj 40 data/$part exp/make_mfcc/$part $mfccdir
  steps/compute_cmvn_stats.sh data/$part exp/make_mfcc/$part $mfccdir
done
```

특징 추출까지의 과정 (run.sh)

● steps/make_mfcc.sh

- Usage
 - steps/make_mfcc.sh [options] <data-dir> <log-dir> <path-to-mfccdir>
- Input parameter
 - [Options]
 - --compress <true|false>: MFCC output의 compress 여부
 - --nj <nj>: MFCC를 추출하기 위해 사용되는 job 개수
(실습에서는 한 사람당 최대 3개의 코어를 사용할 수 있음)
 - --cmd (utils/run.pl | utils/queue.pl <queue opts>)
 - > run.pl ; 단일 서버 상에서 job 수행
 - > queue.pl ; Job queue를 이용하여 다중 서버 상에서 job 수행

특징 추출까지의 과정 (run.sh)

● steps/make_mfcc.sh

- Input parameter
 - <data-dir>
 - MFCC 특징 추출을 위한 wav 파일 목록 (wav.scp) 이 있는 디렉토리
 - <log-dir>
 - MFCC 특징 추출 과정에서의 로그 정보 디렉토리
 - <path-to-mfccdir>
 - 추출된 MFCC 특징 저장 디렉토리

특징 추출까지의 과정 (run.sh)

● steps/compute_cmvn_stats.sh

- Usage
 - `steps/compute_cmvn_stats.sh [options] <data-dir> <log-dir> <path-to-cmvndir>`
- Input parameter
 - [Options]
 - <data-dir>
 - MFCC 특징 추출을 위한 wav 파일 목록 (wav.scp) 이 있는 디렉토리
 - <log-dir>
 - CMVN 추출 과정에서의 로그 정보 디렉토리
 - <path-to-cmvndir>
 - 추출된 CMVN 저장 디렉토리

The Table Concept of Kaldi

- A Table is a collection of objects indexed by a string (string must be nonempty, space-free).
 - E.g. a collection of matrices indexed by utterance-id, representing features.
- “Templates” in C++: e.g. `vector<int>` is a vector of integers. Mechanism for generic code.
- The basic concept is: `Table<Object>`, e.g. `Table<int>`, `Table<Matrix<float> >`
- Handles access to objects on disk (or pipes, etc.)

Tables: form on disk

- Two ways objects are stored on disk:
 - “scp” (script) mechanism: .scp file specifies mapping from key (the string) to filename or pipe:

○○○

```
$ head -2 data/train/wav.scp
trn_adg04_sr009 sph2pipe -f wav /foo/rm1_audio1/rm1/ind_trn/adg0_4/sr009.sph |
trn_adg04_sr049 sph2pipe -f wav /foo/rm1_audio1/rm1/ind_trn/adg0_4/sr049.sph |
```

- “ark” (archive) mechanism: data is all in one file, with utterance id’s (example

○○○

```
$ head -2 data/train/text
trn_adg04_sr009 SHOW THE GRIDLEY+S TRACK IN BRIGHT ORANGE
trn_adg04_sr049 IS DIXON+S LENGTH GREATER THAN THAT OF RANGER
```


Specifying Tables on command line

- ◉ Strings passed from command line say how to read or write Tables.
 - Note: the type of object expected, and whether to read or write, is determined by the program itself.
 - A string interpreted as specifying how to write a table, we call a “wspecifier” in code, etc.
 - A string that specifies how to read a Table is called an “rspecifier”.

Specifying Tables on command line

- ◉ Strings passed from command line say how to read or write Tables.
 - Note: the type of object expected, and whether to read or write, is determined by the program itself.
 - A string interpreted as specifying how to write a table, we call a “wspecifier” in code, etc.
 - A string that specifies how to read a Table is called an “rspecifier”.

Examples of writing Tables

wspecifier	meaning
ark:foo.ark	Write to archive "foo.ark"
scp:foo.scp	Write to files using mapping in foo.scp
ark:-	Write archive to stdout
ark,t:lgzip -c >foo.gz	Write text-form archive to foo.gz
ark,t:-	Write text-form archive to stdout
ark,scp:foo.ark,foo.scp	Write archive and scp file (see below)

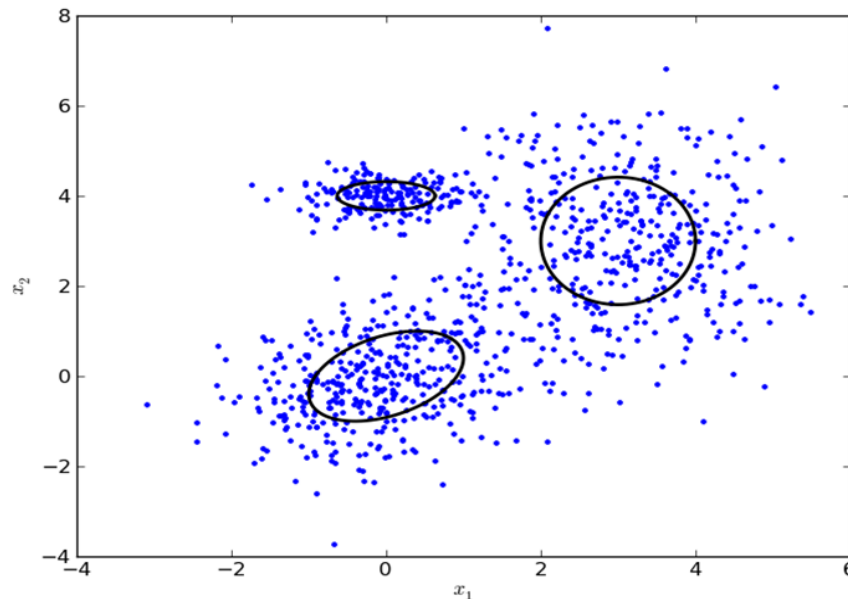
Examples of writing Tables

rspecifier	meaning
ark:foo.ark	Read from archive foo.ark
scp:foo.scp	Read as specified in foo.scp
ark:-	Read archive from stdin
ark:gunzip -c foo.gz	Read archive from foo.gz
ark,s,cs:-	Read archive (sorted) from stdin...

GMM-HMM 기반 음향 모델 학습

● HMM의 state에서 observation 확률을 모델링

- Observation 확률을 Gaussian Mixture Model (GMM)로 표현하여 음향 모델을 학습
- 예) 3개의 gaussian 함수를 이용하여 결합
 - Mean과 variance 파라미터로 구성



GMM-HMM 기반 음향 모델 학습

● GMM-HMM 기반 음향 모델 학습 과정

- Kaldi에서 GMM-HMM 기반 음향 모델 학습 시, 다른 학습 결과물을 이용하는 경우가 많아 순차적으로 음향 모델 학습을 수행해야 함
 - 예) Monophone 음향 모델의 alignment 결과물은 triphone 음향 모델에서 사용

● Monophone 음향 모델 생성 및 alignment

```
steps/train_mono.sh --boost-silence 1.25 --nj 20 --cmd "$train_cmd" \
  data/train_2kshort data/lang_nosp exp/mono
```

```
steps/align_si.sh --boost-silence 1.25 --nj 10 --cmd "$train_cmd" \
  data/train_5k data/lang_nosp exp/mono exp/mono_ali_5k
```

GMM-HMM 기반 음향 모델 학습

● `kaldi-trunk/egs/rm/s5/steps/train_mono.sh`

- Usage
 - `steps/train_mono.sh [options] <data-dir> <lang-dir> <exp-dir>`
- Input parameter
 - [Options]
 - `--nj <nj>`
 - `--cmd (run.pl | queue.pl <queue opts>)`
 - `<data-dir>`
 - 학습 자료 디렉토리
 - `<lang-dir>`
 - WSFT 관련 디렉토리
 - `<exp-dir>`
 - 학습 결과 디렉토리

GMM-HMM 기반 음향 모델 학습

● `kaldi-trunk/egs/rm/s5/steps/train_mono.sh`

- 관련 Kaldi binary 명령어
 - `gmm-init-mono`
 - 모노폰 GMM 모델을 초기화 한다.
 - `compile-train-graphs`
 - 학습 도중에 사용되는 train graph 를 생성한다.
 - `align-equal-compiled`
 - 학습자료에 대한 phoneme sequence align 작업을 수행한다.
 - `gmm-acc-stats-ali`
 - align 결과로부터 gmm training 을 위한 stats 계산을 수행한다.
 - `gmm-est`
 - ML 기반 model re-estimation

GMM-HMM 기반 음향 모델 학습

● `kaldi-trunk/egs/rm/s5/steps/align_si.sh`

- Usage
 - `steps/align_si.sh [options] <data-dir> <lang-dir> <src-dir> <align-dir>`
- Input parameter
 - [Options]
 - <data-dir>
 - 학습 자료 디렉토리
 - <lang-dir>
 - WSFT 관련 디렉토리
 - <src-dir>
 - Align을 수행하려는 디렉토리
 - <align-dir>
 - Alignment 결과 디렉토리

GMM-HMM 기반 음향 모델 학습

● kaldi-trunk/egs/rm/s5/exp/mono

- final.mdl
 - Monophone 음향 모델 파일
 - gmm-info final.mdl ; gmm 모델 정보 출력

```
gmm-info exp/mono/final.mdl
number of phones 346
number of pdfs 127
number of transition-ids 2196
number of transition-states 1058
feature dimension 39
number of gaussians 1003
```

GMM-HMM 기반 음향 모델 학습

● kaldi-trunk/egs/rm/s5/exp/mono

- final.mdl
 - Monophone 음향 모델 파일
 - `gmm-copy --binary=false final.mdl final.mdl.txt`
; Binary 형태로 저장된 final.mdl을 text 형태로 변환

```
Usage: gmm-copy [options] <model-in> <model-out>
```

```
e.g.:
```

```
gmm-copy --binary=false 1.mdl 1_txt.mdl
```

```
Options:
```

```
--binary           : Write output in binary mode (bool, default = true)
--copy-am          : Copy the acoustic model (AmDiagGmm object) (bool, default = true)
--copy-tm          : Copy the transition model (bool, default = true)
```

```
Standard options:
```

```
--config           : Configuration file to read (this option may be repeated) (string, default = "")
--help             : Print out usage message (bool, default = false)
--print-args       : Print the command line arguments (to stderr) (bool, default = true)
--verbose          : Verbose level (higher->more logging) (int, default = 0)
```

GMM-HMM 기반 음향 모델 학습

● steps/train_deltas.sh

- Usage
 - `steps/train_deltas.sh <num-leaves> <tot-gauss> <data-dir> <lang-dir> <alignment-dir> <exp-dir>`
- Input parameter
 - [Options]
 - `<num-leaves>` : HMM의 최종 state의 개수를 지정
 - `<tot-gauss>` : GMM의 전체 가우시안 모델 개수를 지정
 - `<data-dir>`
 - 학습 입력 디렉토리
 - `<lang-dir>`
 - WFST 관련 디렉토리
 - `<exp-dir>`
 - 학습 결과 음향 모델 저장 디렉토리

GMM-HMM 기반 음향 모델 학습

● steps/train_deltas.sh

- 관련 Kaldi binary 명령어
 - acc-tree-stats
 - Phoneme의 문맥적 tree 구성을 위한 통계치를 계산한다.
 - sum-tree-stats
 - tree 구성 통계치 파일을 summation
 - cluster-phones
 - State clustering 이전에 phoneme clustering 수행
 - compile-questions
 - question set txt를 parsing하는 작업
 - build-tree
 - State clustering을 위한 decision tree 구성
- tri 학습 진행시 mono training에서의 binary가 사용됨

GMM-HMM 기반 인식 (디코딩)

● `kaldi-trunk/egs/rm/s5/utils/mkgraph.sh`

- Usage
 - `utils/mkgraph.sh [options] <lang-dir> <model-dir> <graph-dir>`
- Input parameter
 - `<lang-dir>`
 - WSFT 관련 디렉토리
 - `<model-dir>`
 - 음향모델 학습 결과 디렉토리
 - `<graph-dir>`
 - Decoding graph 결과 디렉토리

GMM-HMM 기반 인식 (디코딩)

◉ steps/decode.sh

- Usage
 - steps/decode.sh [options] <graph-dir> <data-dir> <decode-dir>
- Input parameter
 - Options
 - --config
 - Decoding을 수행하기 위한 option들을 포함하고 있는 config 파일의 경로 정보를 설정
 - --nj
 - Parallel job의 개수를 설정
 - --iter
 - Test를 위한 model iteration 횟수를 설정
 - --model
 - Decoding에서 사용할 AM의 경로를 설정

GMM-HMM 기반 인식 (디코딩)

● steps/decode.sh

- Input parameter
 - Options
 - --cmd
 - Job을 실행할 방법에 대해서 설정
 - run.pl ; Job을 단일 서버에서 실행
 - queue.pl ; Job을 다중 서버에서 실행
 - --transform-dir
 - fMLLR transform 파일이 있는 디렉토리 경로 정보
 - --acwt
 - Lattice를 생성하기 위한 AM의 scale 값
 - --scoring-opts
 - Scoring을 수행할 때 추가되는 option 파일의 경로 정보
 - --num-threads
 - Decoding을 수행할 thread의 개수

GMM-HMM 기반 인식 (디코딩)

● steps/decode.sh

- Input parameter
 - Options
 - --parallel-opts
 - Parallel processing을 위한 option 정보를 포함하고 있는 파일의 경로
 - graph-dir
 - Decoding graph에 해당하는 디렉토리 경로
 - data-dir
 - Test corpus에 해당하는 디렉토리 경로
 - decode-dir
 - Decoding 결과를 저장하는 디렉토리 경로

GMM-HMM 기반 인식 (디코딩)

◉ steps/decode.sh

- Output

```
lat.1.gz lat.13.gz lat.17.gz lat.20.gz lat.6.gz log wer_10_0.5 wer_11_1.0
lat.10.gz lat.14.gz lat.18.gz lat.3.gz lat.7.gz num_jobs wer_10_1.0 wer_12_0.0
lat.11.gz lat.15.gz lat.19.gz lat.4.gz lat.8.gz scoring wer_11_0.0 wer_12_0.5
lat.12.gz lat.16.gz lat.2.gz lat.5.gz lat.9.gz wer_10_0.0 wer_11_0.5 wer_12_1.0
```

- wer_* 파일

- *는 decoding 과정에서 적용된 LM weight 값을 의미 (현재 2부터 13까지의 LM weight를 적용하도록 설정됨)
- Decoding 과정에서 LM weight 값에 따라 구한 Word Error Rate

```
compute-wer --text --mode=present ark:exp/tril/decode_nosp_t
%WER 22.51 [ 11833 / 52576, 1283 ins, 1497 del, 9053 sub ]
%SER 88.05 [ 2307 / 2620 ]
Scored 2620 sentences, 0 not present in hyp.
```

GMM-HMM 기반 인식 (디코딩)

● steps/decode.sh

- Output

- num_jobs 파일

- 전체 test 파일을 몇 개의 job으로 나누어서 decoding 했는지를 기록한 파일
 - 만약 20개의 job으로 나누어서 decoding을 수행했다면, num_jobs에는 20이 기록됨

- lat 파일

- Decoding 결과에 대한 lattice 압축 파일

- scoring 디렉토리

- Decoding 과정에서 LM weight을 다르게 적용했을 때의 scoring 값 및 각 LM weight에서의 best decoding path를 기록한 파일들이 저장

- log 디렉토리

- 실제 decoding 과정에서의 인식 결과를 기록한 log 파일들이 저장되어 있음

GMM-HMM 기반 인식 (디코딩)

◉ steps/decode.sh

- Output
 - log 디렉토리
 - log 파일 예시

```
# gmm-latgen-faster --max-active=7000 --beam=13.0 --lattice-beam=6.0 --acoustic-scale=0.083333 --allow-partial=true --word-symbol-t
exp/tril/final.mdl exp/tril/graph_nosp_tgsmall/HCLG.fst "ark,s,cs:apply-cmvn --utt2spk=ark:data/test_clean/split20/1/utt2spk s
ata/test_clean/split20/1/feats.scp ark:- | add-deltas ark:- ark:- |" "ark:|gzip -c > exp/tril/decode_nosp_tgsmall_test_clean/lat
# Started at Thu Jul 20 03:02:09 KST 2017
#
gmm-latgen-faster --max-active=7000 --beam=13.0 --lattice-beam=6.0 --acoustic-scale=0.083333 --allow-partial=true --word-symbol-t
exp/tril/final.mdl exp/tril/graph_nosp_tgsmall/HCLG.fst 'ark,s,cs:apply-cmvn --utt2spk=ark:data/test_clean/split20/1/utt2spk scp
a/test_clean/split20/1/feats.scp ark:- | add-deltas ark:- ark:- |' 'ark:|gzip -c > exp/tril/decode_nosp_tgsmall_test_clean/lat.1
apply-cmvn --utt2spk=ark:data/test_clean/split20/1/utt2spk scp:data/test_clean/split20/1/cmvn.scp scp:data/test_clean/split20/1/f
add-deltas ark:- ark:-
1089-134686-0000 HE HOPED THERE WOULD BE STEW FOR DINNER TURNIPS AND CARROTS AND BRUISED POTATOES AND FAT MUTTON PIECES TO BE LAI
SAUCE
LOG (gmm-latgen-faster[5.1.113~1-72672]:DecodeUtteranceLatticeFaster():decoder-wrappers.cc:286) Log-like per frame for utterance
es.
1089-134686-0001 STUFF IT INTO HIS BELLY COUNSELLED HIM
LOG (gmm-latgen-faster[5.1.113~1-72672]:DecodeUtteranceLatticeFaster():decoder-wrappers.cc:286) Log-like per frame for utterance
s.
1089-134686-0002 AFTER BURNING NIGHTFALL THE ALONE LAMPS WOULD LIGHT UP HERE AND THERE OF THE SQUALID QUARTER OF THE BROTHELS
LOG (gmm-latgen-faster[5.1.113~1-72672]:DecodeUtteranceLatticeFaster():decoder-wrappers.cc:286) Log-like per frame for utterance
s.
```

GMM-HMM 기반 인식 (디코딩)

● gmm-latgen-faster

- 기능
 - GMM 기반 모델을 이용하여 lattice 생성
- Usage
 - `gmm-latgen-faster [options] model-in (fst-in|fst-rspecifier) features-rspecifier lattice-wspecifier`
- Input parameter
 - Options
 - `--max-active`
 - Decoding을 수행할 때 사용되는 최대 active state 개수

GMM-HMM 기반 인식 (디코딩)

● gmm-latgen-faster

- Input parameter
 - Options
 - --beam
 - Decoding 과정에서 적용되는 beam의 크기
 - --lattice-beam
 - Lattice 생성 과정에서 적용되는 beam의 크기
 - --acoustic-scale
 - AM의 likelihood를 위한 scaling factor
 - --allow-partial
 - Decoding 과정에서 end state에 도달하지 못할 경우, 이전의 state까지만을 고려했을 때의 decoding 결과를 lattice로 저장할지의 여부
 - --word-symbol-table
 - Lexicon에 포함된 word에 대한 symbol table

GMM-HMM 기반 인식 (디코딩)

● gmm-latgen-faster

- Input parameter
 - model-in
 - AM이 저장된 경로 정보
 - fst-in|fst-specifier
 - FST 타입의 decoding graph가 저장된 경로
 - features-specifier
 - AM의 feature가 저장된 경로
 - lattice-specifier
 - 생성된 lattice를 저장하기 위한 lattice 파일 이름 정보
- Output
 - GMM 기반 모델을 통해서 decoding한 lattice 파일

인식 결과 확인 과정

- Kaldi의 scoring 과정은 Decoding script에서 함께 수행됨
 - steps/decode.sh의 최종 단계로 scoring을 수행하는 쉘 스크립트를 수행
- Decoding의 결과물인 lattice 파일들을 이용하여 scoring을 계산
 - Scoring을 위한 전제 조건으로 lattice 파일들이 미리 생성되었다고 가정

인식 결과 확인 과정

● lattice 확인 과정

- `lattice-copy "ark:gunzip -c lat.1.gz|" ark,t:- | utils/int2sym.pl -f 3 data/lang/words.txt | less`

● lattice visualization

- `utils/show_lattice.sh 1089-134686-0000 "exp/tri4b/decode_tgsmall_test_clean/lat.*.gz" exp/tri4b/graph_tgsmall/words.txt`

인식 결과 확인 과정

● local/score.sh

- Usage
 - local/score.sh [options] <data-dir> <lang-dir|graph-dir> <decode-dir>
- Input parameter
 - Options
 - --cmd
 - Job을 실행할 방법에 대해서 설정
 - run.pl ; Job을 단일 서버에서 실행
 - queue.pl ; Job을 다중 서버에서 실행
 - --min_lmwt
 - Lattice rescoring을 위한 최소 LM weight 값
 - --max_lmwt
 - Lattice rescoring을 위한 최대 LM weight 값
- data-dir
 - Test data가 위치한 경로

인식 결과 확인 과정

● local/score.sh

- Input parameter
 - lang-dir|graph-dir
 - Lang 디렉토리 또는 decoding graph가 저장된 디렉토리 경로
 - decode-dir
 - decode 결과물이 저장된 디렉토리 경로
- Output
 - decode 디렉토리의 서브 디렉토리인 scoring 디렉토리 생성

GMM-HMM 기반 음향 모델 학습

- Linear Discriminant Analysis (LDA) 및 Maximum Likelihood Linear Transform (MLLT) 음향 모델 생성 및 alignment

```
# train an LDA+MLLT system.  
steps/train_lda_mllt.sh --cmd "$train_cmd" \  
  --splice-opts "--left-context=3 --right-context=3" 2500 15000 \  
  data/train_10k data/lang_nosp exp/tri1_ali_10k exp/tri2b
```

- Triphone 음향 모델에 대해 LDA transform과 MLLT estimation 수행

GMM-HMM 기반 음향 모델 학습

- Speaker Adapted Training (SAT) 음향 모델 생성 및 alignment

```
# Train tri3b, which is LDA+MLLT+SAT on 10k utts
steps/train_sat.sh --cmd "$train_cmd" 2500 15000 \
  data/train_10k data/lang_nosp exp/tri2b.ali_10k exp/tri3b
```

- steps/train_sat.sh에서 speaker adapted 특징 정보인 fMLLR이 없어도 alignment된 정보를 이용하여 수행

GMM-HMM 기반 음향 모델 학습

● `kaldi-trunk/egs/rm/s5/steps/train_sat.sh`

- Usage

- `steps/train_sat.sh [options] <num-leaves> <tot-gauss> <data-dir> <lang-dir> <alignment-dir> <exp-dir>`

● `kaldi-trunk/egs/rm/s5/steps/align_fmllr.sh`

- Usage

- `steps/align_fmllr.sh [options] <data-dir> <lang-dir> <src-dir> <align-dir>`

- Input parameter

- [Options]

- `--use-graphs true`
 - `<src-dir>`의 음향모델에 대한 fst 정보를 사용한다는 의미
 - False로 설정할 수 있지만 fst 정보를 사용했을 때, 성능이 더 좋다고 기술되어 있음

SRILM을 이용한 언어모델 학습

● Stanford Research Institute Language Model (SRILM) toolkit

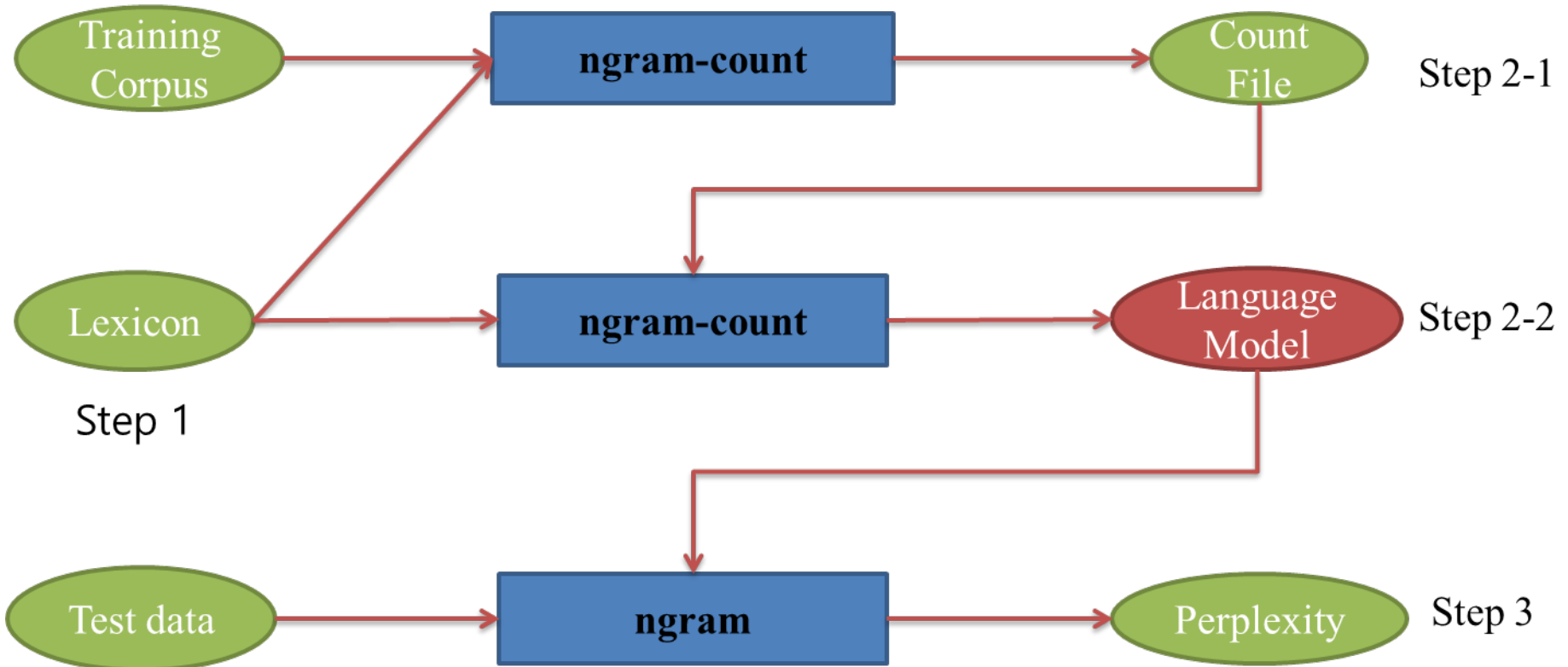
- 목적
 - 통계기반 언어모델 생성 및 평가를 위한 toolkit
- 적용 분야
 - 음성인식
 - 기계 번역
 - 자연어 처리

● Reference site

- <http://www-speech.sri.com/projects/srilm/>

SRILM을 이용한 언어모델 학습

- n-gram 언어모델의 ARPA format을 생성하기 위해 SRILM toolkit 사용



SRILM을 이용한 언어모델 학습

● ngram-count 를 이용한 vocabulary 생성하기

• Step 1 (Lexicon or Vocabulary)

• Usage

- (예) `ngram-count -text training.txt -write-vocab training.vocab -write1 unigram.freq`

• Input Parameter

- `-text`
 - 언어모델 학습 자료 코퍼스
- `-write-vocab`
 - 생성할 어휘 사전 출력 파일
- `-write1`
 - Unigram 빈도수 정보 출력 파일

SRILM을 이용한 언어모델 학습

◉ ngram-count 를 이용한 vocabulary 생성하기

- Step 1 (Lexicon or Vocabulary) 결과
 - training.vocab 파일

```
-pau-  
0  
010  
1  
10  
1000점  
1000점은  
1000점이  
1000점이나  
100206  
100만대의  
10년  
10명의
```

SRILM을 이용한 언어모델 학습

◉ ngram-count 를 이용한 vocabulary 생성하기

- Step 1 (Lexicon or Vocabulary) 결과
 - unigram.freq 파일

맏음으로써	1	
만들어주실래나요		1
영역	2	
구조입니다	1	
나서고	1	
영화처럼	1	
정치가가	1	
영향을	1	
엠엔프라이스	1	
회사와의	1	

SRILM을 이용한 언어모델 학습

● ngram-count 를 이용한 빈도수 통계정보 추출

• Step 2-1 (빈도수 통계정보 추출)

• Usage

- (예) `ngram-count -vocab training.vocab -text training.txt -order 3 -write training.count`

• Input Parameter

- `-vocab`
 - 어휘 사전 파일
- `-text`
 - 언어모델 학습 자료 코퍼스
- `-order`
 - n-gram의 n 설정
- `-write`
 - n-gram별 빈도수 정보 출력 파일

SRILM을 이용한 언어모델 학습

● ngram-count 를 이용한 빈도수 통계정보 추출

- Step 2-1 (빈도수 통계정보 추출) 결과
 - training.count 파일

```
선택요금할인제도가      1
선택요금할인제도가 없나요      1
선택요금할인제도가 없나요 </s> 1
하기까지      1
하기까지 하면서      1
하기까지 하면서 말이다      1
덜었군요      1
덜었군요 </s>      1
```

SRILM을 이용한 언어모델 학습

● ngram-count를 이용한 언어모델 생성

• Step 2-2 (언어모델 생성)

• Usage

- (예) `ngram-count -vocab training.vocab -read training.count -order 3 -lm training.lm`

• Input Parameter

- `-vocab`
 - 어휘 사전 파일
- `-order`
 - n-gram의 n 설정
- `-read`
 - n-gram별 빈도수 정보 파일
- `-lm`
 - 생성할 언어모델 출력 파일

SRILM을 이용한 언어모델 학습

◉ ngram-count를 이용한 언어모델 생성

- Step 2-2 (언어모델 생성) 결과
 - training.lm 파일

```
\data\  
ngram 1=3659  
ngram 2=5603  
ngram 3=46  
  
\1-grams:  
-3.783475      0      0.05077268  
-3.783475      010     -0.02728014  
-2.607383      1      0.03790811  
-3.783475      10     -0.02728014  
-2.829232      1000점  0.02579107  
-3.783475      1000점은 -0.02728014  
-3.783475      1000점이 -0.02728014  
-3.783475      1000점이나 -0.02720862  
-3.783475      100206  0.05077268
```

SRILM을 이용한 언어모델 학습

● ngram을 이용한 언어모델 성능 평가

• Step 3 (언어모델 성능 평가 (Perplexity))

• Usage

- (예) `ngram -vocab training.vocab -order 3 -lm training.lm -ppl test.txt -debug 2`

• Input Parameter

- `-lm`
 - 언어모델 파일
- `-ppl`
 - 성능 측정할 테스트 파일
- `-debug 2`
 - 문장별 로그 정보 출력 설정 옵션

SRILM을 이용한 언어모델 학습

- ngram을 이용한 언어모델 성능 평가
 - Step 3 (언어모델 성능 평가 (Perplexity))결과

```
reading 3659 1-grams
reading 5603 2-grams
reading 46 3-grams
홈 줌 까다롭네요 그래서 며칠전 제 글이 조용히 사라졌군요
  p( 홈 | <s> )      = [2gram] 6.10372e-05 [ -4.21441 ]
  p( 줌 | 홈 ...)    = [2gram] 0.0610373 [ -1.2144 ]
  p( 까다롭네요 | 줌 ...) = [2gram] 0.0101729 [ -1.99256 ]
  p( 그래서 | 까다롭네요 ...) = [2gram] 0.0610373 [ -1.2144 ]
  p( 며칠전 | 그래서 ...) = [2gram] 0.0122075 [ -1.91338 ]
  p( 제 | 며칠전 ...) = [2gram] 0.0610373 [ -1.2144 ]
  p( 글이 | 제 ...) = [2gram] 0.00871961 [ -2.0595 ]
  p( 조용히 | 글이 ...) = [2gram] 0.0610373 [ -1.2144 ]
  p( 사라졌군요 | 조용히 ...) = [2gram] 0.0305186 [ -1.51543 ]
  p( </s> | 사라졌군요 ...) = [2gram] 0.0610373 [ -1.2144 ]
1 sentences, 9 words, 0 OOVs
0 zeroprobs, logprob= -17.7673 ppl= 59.804 ppl1= 94.2203
```

DNN-HMM 기반 음향 모델 학습

● Kaldi의 DNN-HMM 기반 음향 모델

- nnet
- nnet2
- nnet3

DNN-HMM 기반 음향 모델 학습

● nnet

- Karel의 학습 방법 (kaldi-trunk/egs/wsj/s5/local/nnet)
- Single CPU core 또는 single GPU 사용
- Clustering된 서버 상에서 multiple GPU 사용 가능
- Validation data를 사용하여 early stopping 수행
- 학습된 DNN-HMM 기반 음향 모델의 성능이 수렴하지 못할 수도 있음
- 다양한 DNN-HMM 기반 음향 모델 학습 제공
 - FFNN
 - DBN
 - CNN
 - LSTM RNN
 - Bi-directional LSTM RNN

DNN-HMM 기반 음향 모델 학습

● nnet2

- Povey의 학습 방법 (kaldi-trunk/egs/wsj/s5/local/nnet2)
- Multiple CPU core 또는 multiple GPU 사용
- GridEngine 상의 다중 서버들의 CPU 또는 GPU를 사용
- Data parallelism에 기반한 모델 학습 수행
 - Natural gradient
 - Parameter averaging
- 정해진 횟수의 epoch을 수행
- Epoch의 횟수에 따른 DNN-HMM 기반 음향 모델의 성능을 파악하기 쉬움
- FFNN을 이용한 DNN-HMM 기반 음향 모델 학습 제공

DNN-HMM 기반 음향 모델 학습

● nnet3

- nnet3 학습 방법 (kaldi-trunk/egs/wsj/s5/steps/nnet3)
- Multiple CPU core 또는 multiple GPU 사용
- GridEngine 상의 다중 서버들의 CPU 또는 GPU를 사용
- Data parallelism에 기반한 모델 학습 수행
 - Natural gradient
 - Parameter averaging
- 사용자가 정의한 DNN-HMM 모델을 script level에서 구현 가능
- Python 기반의 script 파일을 수정하여 사용자가 정의한 DNN-HMM 모델을 구현 가능
- LSTM / TDNN 등 최근 연구되고 있는 모델 반영

DNN-HMM 기반 음향 모델 학습

● steps/nnet2/train_pnorm_fast.sh

- Usage
 - `steps/nnet2/train_pnorm_fast.sh [options] <data-dir> <lang-dir>`
`<alignment-dir> <exp-dir>`
- Input parameter
 - [Options]
 - `--stage`: `train_pnorm_fast.sh` 내부는 각 단계별로 작업을 수행할 수 있도록 script가 구성되어 있으며 이 option을 통해 수행하고자 하는 단계만 수행 가능
 - `--num-epochs`: DNN 모델 학습 시 learning rate를 감소시키기 위한 epoch의 횟수
 - `--num-epochs-extra`: `--num-epochs` 만큼 epoch을 수행한 이후에 추가로 수행하는 epoch의 횟수
(전체 epoch 횟수: $(--num-epochs) + (--num-epochs-extra)$)

DNN-HMM 기반 음향 모델 학습

● local/nnet2/run_5a_clean_100.sh

- Usage
 - steps/nnet2/train_pnorm_fast.sh를 이용한 DNN-HMM 음향 모델 학습 sh 스크립트
 - GPU 사용 설정: GPU 사용시 NVCC 설치 여부를 확인

```
if $use_gpu; then
    if ! cuda-compiled; then
        cat <<EOF && exit 1
This script is intended to be used with GPUs but you have not compiled Kaldi with CUDA
If you want to use GPUs (and have them), go to src/, and configure and make on a machine
where "nvcc" is installed.  Otherwise, call this script with --use-gpu false
EOF
    fi
    num_threads=1
    minibatch_size=512
    # the _a is in case I want to change the parameters.
    dir=exp/nnet2_online/nnet_a_gpu_baseline
else
    # Use 4 nnet jobs just like run_4d_gpu.sh so the results should be
    # almost the same, but this may be a little bit slow.
    num_threads=16
    minibatch_size=128
    dir=exp/nnet2_online/nnet_a_baseline
fi
```

DNN-HMM 기반 음향 모델 학습

● `kaldi-trunk/egs/wsj/s5/steps/nnet2/train_pnorm_fast.sh`

- Input parameter

- [Options]

- `--splice-width N`: 현재 input frame을 기준으로 앞으로 N개, 뒤로 N개의 frame을 고려 (이전의 context option의 기능과 동일)
 - `--cmvn-opts: steps/nnet2/get_lda.sh`로 전달하는 option으로 cmvn에 대한 option을 제공
 - `--num-threads`: DNN 음향 모델 학습 시 사용되는 thread의 개수
 - `--minibatch-size`: Minibatch의 크기
 - Kaldi에서는 $(--num-threads) \times (--minibatch-size)$ 값이 2,000보다 크지 않는 것을 권장 (성능 저하 발생)
 - `--parallel-opts: run.pl` 또는 `queue.pl`에서 사용하기 위한 option
 - `--num-jobs-nnet`: 동시에 수행하기 위한 job의 개수
 - Kaldi에서는 `--num-jobs-nnet` 값을 증가시켰을 때, learning rate도 증가시키는 것을 권장

DNN-HMM 기반 음향 모델 학습

● `kaldi-trunk/egs/wsj/s5/steps/nnet2/train_pnorm_fast.sh`

- Input parameter

- [Options]

- `--num-hidden-layers`: DNN 모델의 hidden layer의 개수
 - Kaldi에서는 hidden layer로 3~10시간 분량의 학습자료의 경우 2개,
100~500 시간 분량의 학습자료의 경우 4개,
1000시간 이상의 학습자료의 경우 5개를 사용하는 것을 권장
 - `--mix-up`: Softmax까지 수행한 DNN 모델의 output posterior의 개수
 - 현재 Kaldi에서는 이 option을 더 이상 지원하지 않음
 - `--initial-learning-rate`: 초기 learning rate 값
 - Kaldi에서는 소용량 학습 자료의 경우 0.02, 대용량 학습 자료의 경우 0.01로 설정하는 것을 권장
 - `--final-learning-rate`: 학습이 끝났을 때의 learning rate 값
 - Kaldi에서는 소용량 학습 자료의 경우 0.004, 대용량 학습 자료의 경우 0.001로 설정하는 것을 권장

DNN-HMM 기반 음향 모델 학습

- `kaldi-trunk/egs/wsj/s5/steps/nnet2/train_pnorm_fast.sh`
 - Input parameter
 - [Options]
 - `--pnorm-input-dim`: p-norm nonlinearity function의 input dimension
 - `--pnorm-output-dim`: p-norm nonlinearity function의 output dimension

DNN-HMM 기반 음향 모델 학습

● 학습된 DNN-HMM 모델 정보

- nnet-am-info final.mdl

```
nnet-am-info final.mdl
num-components 17
num-updatable-components 5
left-context 5
right-context 5
input-dim 40
output-dim 6425
parameter-dim 6496000
```

DNN-HMM 기반 음향 모델 학습

● 학습된 DNN-HMM 모델 정보

- nnet-am-copy -binary=false final.mdl

```
<TransitionModel>
<Topology>
<TopologyEntry>
<ForPhones>
6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64
65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 9
3 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 11
6 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137
138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157
</ForPhones>
<State> 0 <PdfClass> 0 <Transition> 0 0.75 <Transition> 1 0.25 </State>
<State> 1 <PdfClass> 1 <Transition> 1 0.75 <Transition> 2 0.25 </State>
<State> 2 <PdfClass> 2 <Transition> 2 0.75 <Transition> 3 0.25 </State>
<State> 3 </State>
</TopologyEntry>
<TopologyEntry>
<ForPhones>
1 2 3 4 5
</ForPhones>
<State> 0 <PdfClass> 0 <Transition> 0 0.25 <Transition> 1 0.25 <Transition> 2 0.25 <Tr
ansition> 3 0.25 </State>
<State> 1 <PdfClass> 1 <Transition> 1 0.25 <Transition> 2 0.25 <Transition> 3 0.25 <Tr
ansition> 4 0.25 </State>
<State> 2 <PdfClass> 2 <Transition> 1 0.25 <Transition> 2 0.25 <Transition> 3 0.25 <Tr
ansition> 4 0.25 </State>
<State> 3 <PdfClass> 3 <Transition> 1 0.25 <Transition> 2 0.25 <Transition> 3 0.25 <Tr
ansition> 4 0.25 </State>
<State> 4 <PdfClass> 4 <Transition> 4 0.75 <Transition> 5 0.25 </State>
<State> 5 </State>
</TopologyEntry>
</Topology>
<Triples> 21452
1 0 0
```

openfst를 이용한 HCLG 생성

Geeky Stuff

11 Decoding graph construction in Kaldi:

JUN 2012

A visual walkthrough

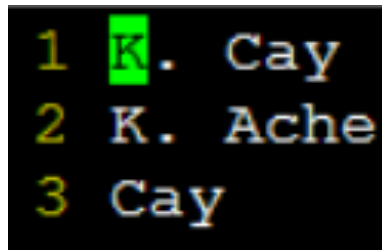
I've got bitten recently by an issue in a Kaldi decoding cascade I was working on. I was getting more than 40% WER, while the language and acoustic model I was using suggested the decoder should have been able to do much better than that. After a lot of head-scratching I've found the reason for the sub-optimal performance was that I simply failed to add self-loops to the lexicon transducer(L). These are needed in order to pass through the special "#0" symbol used in Kaldi's recipes to make the grammar transducer(G) determinizable. The effect of this omission was that the back-off arcs in the bigram G I was using were effectively cut-off, leading to a highly non-stochastic LG cascade with a very spiky distribution over the allowed word sequences and hence the higher WER. After adding the self-loop the WER went down to 17% without changing anything else.

- 출처: <http://vpanayotov.blogspot.kr/2012/06/kaldi-decoding-graph-construction.html>

openfst를 이용한 HCLG 생성

● training.txt

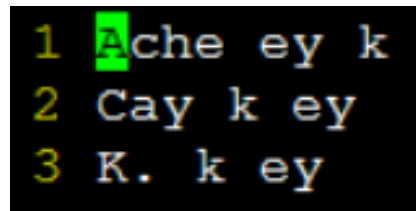
- Text corpus for toy problem



A screenshot of a text file named training.txt. It contains three lines of text, each preceded by a yellow number. The first line is '1 K. Cay', the second is '2 K. Ache', and the third is '3 Cay'. The letter 'K' in the first line is highlighted with a green square.

```
1 K. Cay
2 K. Ache
3 Cay
```

● lexicon.txt



A screenshot of a text file named lexicon.txt. It contains three lines of text, each preceded by a yellow number. The first line is '1 Ache ey k', the second is '2 Cay k ey', and the third is '3 K. k ey'. The letter 'A' in the first line is highlighted with a green square.

```
1 Ache ey k
2 Cay k ey
3 K. k ey
```

openfst를 이용한 HCLG 생성

● tri.mdl

- kaldi format acoustic model file

```
gmm-info tri2a.mdl
number of phones 42
number of pdfs 1471
number of transition-ids 3098
number of transition-states 1545
feature dimension 39
number of gaussians 9018
```

● tree

- kaldi state clustering decision tree

openfst를 이용한 HCLG 생성

● tids.txt

- State idx list


```
<eps> 0
aa_0_125_0 1
aa_0_125_1 2
aa_0_284_0 3
aa_0_284_1 4
aa_0_310_0 5
aa_0_310_1 6
aa_0_345_0 7
aa_0_345_1 8
aa_0_396_0 9
aa_0_396_1 10
aa_0_430_0 11
```


openfst를 이용한 HCLG 생성

⦿ Grammar Transducer 생성

(G.fst)

- srilm toolkit을 이용한 lm 생성

```
1   
2 \data\  
3 ngram 1=5  
4 ngram 2=6  
5 ngram 3=0  
6  
7 \1-grams:  
8 -0.4259687 </s>  
9 -99 <s> -0.30103  
10 -0.90309    Ache    -0.09691  
11 -0.60206    Cay    -0.2730013  
12 -0.60206    K.     -0.2730013  
13  
14 \2-grams:  
15 -0.60206    <s> Cay  
16 -0.30103    <s> K.  
17 -0.30103    Ache </s>  
18 -0.1760913  Cay </s>  
19 -0.4771213  K. Ache  
20 -0.4771213  K. Cay  
21  
22 \3-grams:  
23  
24 \end\
```

openfst를 이용한 HCLG 생성

● Grammar Transducer 생성

(G.fst)

- `cat training.lm | arpa2fst - | fstprint | eps2disambig.pl | s2eps.pl | fstcompile --isymbols=words.txt --osymbols=words.txt --keep_isymbols=false --keep_osymbols=false | fstrmepsilon > G.fst`
- `arpa2fst`: arpa format language model을 fst 포맷으로 변환
- `fstprint`: fst binary 포맷을 txt 형태로 출력
- `eps2disambig.pl`: txt형태이 fst에서 word 사이의 `<eps>`를 disambiguation symbol로 변환
- `s2eps.pl`: `<s>` 및 `</s>` symbol을 `<eps>`로 변환

openfst를 이용한 HCLG 생성

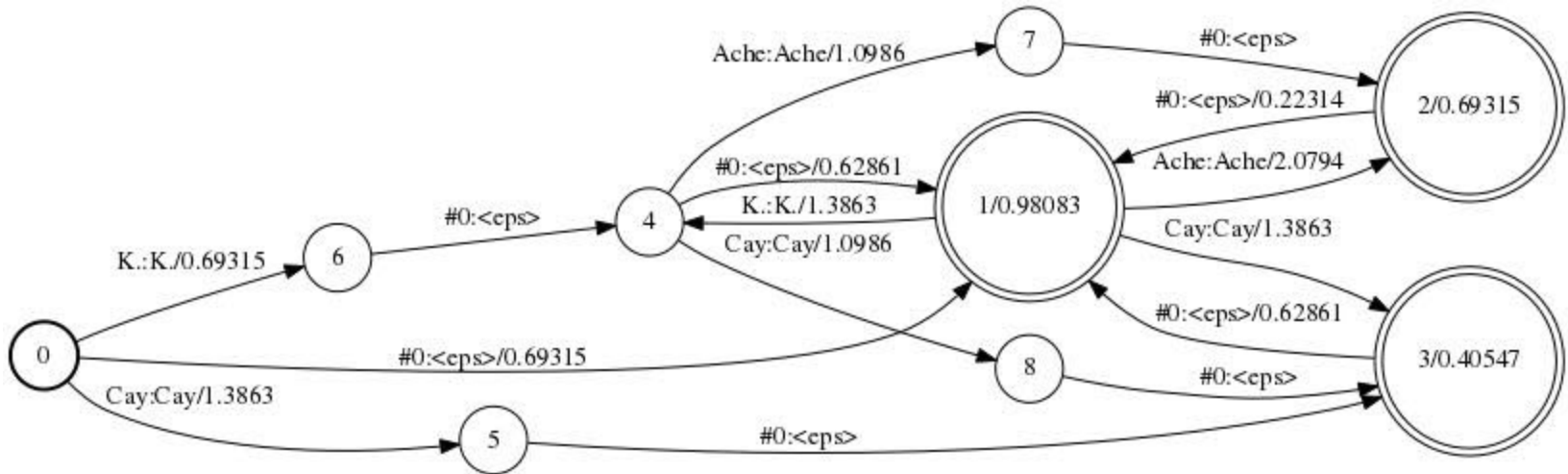
⦿ Grammar Transducer 생성

(G.fst)

- `cat training.lm | arpa2fst - | fstprint | eps2disambig.pl | s2eps.pl | fstcompile --isymbols=words.txt --osymbols=words.txt --keep_isymbols=false --keep_osymbols=false | fstrmepsilon > G.fst`
- `fstcompile`: txt형태의 fst 포맷을 binary형태의 fst로 변환
- `fstrmepsilon`: fst 상에서 <eps>를 최소화

openfst를 이용한 HCLG 생성

- fst 파일을 jpg 로 변환
 - `fstdraw G.fst | dot -Tjpg > G.jpg`



openfst를 이용한 HCLG 생성

● Lexicon 생성 (L.fst)

- `add_lex_disambig.pl lexicon.txt lexicon_disambig.txt`

```
1 Ache ey k
2 Cay k ey #1
3 K. k ey #2
```

openfst를 이용한 HCLG 생성

● Lexicon 생성 (L.fst)

- `make_lexicon_fst.pl lexicon.txt 0.5 sil | fstcompile --
isymbols=lexgraphs/phones.txt --osymbols=lmgraphs/
words.txt --keep_isymbols=false --keep_osymbols=false |
fstarcsort --sort_type=olabel > lexgraphs/L.fst`

openfst를 이용한 HCLG 생성

◉ Disambiguation lexicon 생성 (L_disambig.fst)

```
phone_disambig_symbol=`grep \#0 lexgraphs/phones_disambig.txt | awk '{print $2}'`  
word_disambig_symbol=`grep \#0 lmgraphs/words.txt | awk '{print $2}'`  
make_lexicon_fst.pl lexgraphs/lexicon_disambig.txt $silprob sil '#'$ndisambig | \  
  fstcompile --isymbols=lexgraphs/phones_disambig.txt \  
    --osymbols=lmgraphs/words.txt \  
    --keep_isymbols=false --keep_osymbols=false |\  
  fstaddselfloops "echo $phone_disambig_symbol |" "echo $word_disambig_symbol |" | \  
  fstarcsort --sort_type=olabel \  
  > lexgraphs/L_disambig.fst  
  
cat lexgraphs/L_disambig.fst |\  
  fstdraw -isymbols=lexgraphs/phones_disambig.txt -osymbols=lmgraphs/words.txt -portrait=true |\  
  dot -Tpdf >lexgraphs/L_disambig.pdf
```

openfst를 이용한 HCLG 생성

● LG.fst composition

- `fsttablecompose lexgraphs/L_disambig.fst lmgraphs/G_bi.fst |
fstdeterminizestar --use-log=true | fstminimizeencoded > cascade/
LG_bi.fst`
- `fsttablecompose`: fst1 과 fst2에 대한 composition algorithm 수행
- `fstdeterminizestar`: fst에 대한 lattice determinize
(rm epsilon와 determinize를 동시에 수행)
- `fstminimizeencoded`: Encoding 이후에 fst minimize를 수행

openfst를 이용한 HCLG 생성

- Context-dependency transducer (C.fst) 생성

```
fstmakecontextfst --read-disambig-syms=lexgraphs/disambig_phones.list \  
  --write-disambig-syms=cdgraphs/disambig_ilabels.list $phones $subseq_sym \  
  cdgraphs/ilabels | fstarcsort --sort_type=olabel > cdgraphs/C.fst  
  
# Make context-dependent symbols for rendering  
fstmakecontextsyms lexgraphs/phones.txt cdgraphs/ilabels > cdgraphs/ctxsyms.txt  
  
# Render C  
fstdraw -portrait=true \  
  -isymbols=cdgraphs/ctxsyms.txt -osymbols=cdgraphs/phones_disambig_subseq.txt \  
  < cdgraphs/C.fst | dot -Tpdf > cdgraphs/C.pdf
```

openfst를 이용한 HCLG 생성

- CLG cascade (CLG.fst) 생성

```
fsttablecompose cdgraphs/C.fst cdgraphs/LG_bi_subseq.fst > cascade/CLG_bi.fst

fstdraw -portrait=true \
  -isymbols=cdgraphs/ctxsyms.txt \
  -osymbols=lmgraphs/words.txt \
  cascade/CLG_bi.fst | dot -Tpdf > cascade/CLG_bi.pdf
```

openfst를 이용한 HCLG 생성

- HMM state transducer (H.fst) 생성

```
make-ilabel-transducer --write-disambig-syms=cdgraphs/disambig_ilabels_remapped.list\  
  cdgraphs/ilabels $tree $model cdgraphs/ilabels.remapped \  
> cdgraphs/ilabel_map.fst  
  
# Make a symbol table for visualization of the physical C input symbols  
fstmakecontextsyms lexgraphs/phones.txt cdgraphs/ilabels.remapped \  
> cdgraphs/ctxsyms.remapped.txt
```

```
make-h-transducer --disambig-syms-out=hmmgraphs/disambig_tstate.list \  
  --transition-scale=1.0 cdgraphs/ilabels.remapped \  
  $tree $model > hmmgraphs/Ha.fst
```

openfst를 이용한 HCLG 생성

- WFST decoding graph (HCLG.fst) 생성

```
fsttablecompose hmmgraphs/Ha.fst cascade/CLG_bi_reduced.fst | \  
  fstdeterminizestar --use-log=true | \  
  fstrmsymbols hmmgraphs/disambig_tstate.list | \  
  fstrmepslocal | fstminimizeencoded \  
  > cascade/HCLGa_bi.fst  
  
fstdraw -portrait=true \  
  -isymbols=hmmgraphs/tids.txt \  
  -osymbols=lmgraphs/words.txt cascade/HCLGa_bi.fst |\  
dot -Tpdf > cascade/HCLGa_bi.pdf  
  
echo "- Adding the self-loops to HCLGa_uni -> cascade/HCLG_bi.fst"  
add-self-loops --self-loop-scale=0.1 \  
  --reorder=true $model < cascade/HCLGa_bi.fst > cascade/HCLG_bi.fst
```